Titre: Title:	Méthode de reconfiguration dynamique pour un réseau-sur-puce tolérant aux fautes
Auteur: Author:	Jessica Allard Bernier
Date:	2011
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Allard Bernier, J. (2011). Méthode de reconfiguration dynamique pour un réseau-sur-puce tolérant aux fautes [Mémoire de maîtrise, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/745/

URL de PolyPublie: PolyPublie URL:	https://publications.polymtl.ca/745/
Directeurs de recherche: Advisors:	Guy Bois
Programme: Program:	Génie informatique

UNIVERSITÉ DE MONTRÉAL

MÉTHODE DE RECONFIGURATION DYNAMIQUE POUR UN RÉSEAU-SUR-PUCE TOLÉRANT AUX FAUTES

JESSICA ALLARD BERNIER

DÉPARTEMENT DE GÉNIE INFORMATIQUE ET DE GÉNIE LOGICIEL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION

DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES

(GÉNIE INFORMATIQUE)

DÉCEMBRE 2011

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé:

MÉTHODE DE RECONFIGURATION DYNAMIQUE POUR UN RÉSEAU-SUR-PUCE TOLÉRANT AUX FAUTES

présenté par : ALLARD BERNIER Jessica

en vue de l'obtention du diplôme de : Maîtrise ès Sciences Appliquées
a été dûment accepté par le jury d'examen constitué de :

M. LANGLOIS Pierre J.M., Ph. D., président

M. BOIS Guy, Ph. D., membre et directeur de recherche

M. LANGEVIN Michel, Ph. D., membre

REMERCIEMENTS

Je tiens tout d'abord à remercier mon directeur de recherche, M. Guy Bois, pour son support et son encadrement tout au long de ma maîtrise, mais aussi pour la confiance qu'il m'a accordée en me permettant de réaliser ce projet.

Je tiens également à remercier STMicroelectronics pour leur intérêt face à ce projet et pour l'aide qu'ils m'ont offerte au niveau de la définition de mon projet de recherche.

Mes remerciements vont également à toute l'équipe du GRM (Groupe de recherche en microélectronique), notamment à Réjean Lepage, administrateur réseau, et Jean Bouchard, technicien, pour leur soutien technique et leur disponibilité.

Un merci particulier à Hubert Guérard, collègue du laboratoire CIRCUS, pour l'aide et les conseils qu'il a su m'apporter au cours de ces dernières années. Également, merci à Jérôme Collin, chargé d'enseignement au département de Génie informatique et Génie logiciel, pour les petits trucs qu'il a su partager avec moi et toutes les fois où il a pu me dépanner.

Je remercie la CMC pour avoir fourni les outils nécessaires au bon déroulement de ce projet.

D'un côté un peu plus personnel, je remercie mes parents de toujours avoir cru en moi. Je veux finalement remercier mon copain, Eric, qui a su m'épauler et me motiver pour mener à terme ce projet, mais aussi pour toutes les pages qu'il a lues et relues et ses commentaires constructifs.

Merci à tous !

RÉSUMÉ

Avec l'avancement des technologies, les systèmes intégrés ne cessent de gagner en complexité dans le but de remplir toujours plus de fonctionnalités. Pour augmenter les performances et permettre de connecter plus de ressources ensemble pour répondre aux nouveaux besoins sans dégrader les communications, l'utilisation des réseaux sur puce a fait son apparition. Beaucoup d'architectures de réseaux sur puce ont été proposées lors des dernières années. Elles diffèrent selon leur topologie, leur technique de commutation, leurs interconnexions, l'implémentation de leurs nœuds, etc. Parmi les nouvelles architectures de réseaux sur puce, nous comptons notamment le Rotator-on-Chip (RoC) qui a été développé en partenariat par l'École Polytechnique de Montréal et STMicroelectronics.

La constante miniaturisation des transistors a mené les réseaux sur puce, et même tous les systèmes, à devenir de plus en plus vulnérables aux fautes. Il est devenu essentiel pour rencontrer un certain niveau de fiabilité d'intégrer des méthodes de tolérance aux fautes dans les systèmes. Les fautes peuvent affecter les circuits de différentes façons. Elles peuvent avoir des effets permanents comme des défauts de fabrication par exemple. Elles peuvent également engendrer de mauvais fonctionnements ayant un comportement transitoire comme dans le cas d'interférences magnétiques.

L'objectif est que les systèmes remplissent leurs fonctionnalités même en présence de fautes. Dans le cas des réseaux sur puce, la fonctionnalité principale est de transmettre des messages entre les ressources. Alors, le but est de s'assurer qu'aucun bit ne soit altéré lors du transfert. Pour ce faire, plusieurs recherches ont déjà été menées à ce sujet, exploitant des concepts de détection, de redondance, de retransmission, de correction, etc.

Ce mémoire met tout d'abord en lumière le concept de réseau sur puce. Il définit les différentes bases de leur implémentation, notamment leurs composants, leurs techniques de commutation et de routage et leurs topologies. Puisque la tolérance aux fautes est un aspect central de ce travail, une explication de la provenance et de la classification des fautes pouvant survenir à l'intérieur des réseaux sur puce est également détaillée. Pour parvenir à contrer l'effet de ces fautes, plusieurs techniques de tolérance aux fautes ont été développées, que ce soit par l'industrie ou suite à des recherches universitaires. Ce travail présente quelques-unes d'entre elles.

L'essentiel de ce mémoire repose sur l'élaboration d'une technique de tolérance aux fautes adaptée aux caractéristiques du RoC. La technique proposée est de type prévention plutôt que de type correction. Lorsqu'une faute est détectée sur le chemin de données, la méthode s'assure que la faute n'affectera pas d'autres paquets en reconfigurant les canaux du RoC. Pour parvenir à définir la technique de prévention la plus efficace, plusieurs implémentations sont comparées. La couche où la technique de prévention est gérée, que ce soit au niveau logiciel ou matériel, le mécanisme de détection utilisé et la méthode de reconfiguration des canaux font notamment partie de celles-ci.

Pour déterminer quelle technique de prévention s'avère être la plus efficace pour le RoC, ces dernières sont évaluées en termes du nombre d'erreurs évitées, du délai moyen des paquets, du temps de vérification nécessaire et du nombre de ressources utilisées. Les simulations effectuées démontrent qu'une technique intégrée au niveau matériel nécessite beaucoup plus de ressources qu'une technique gérée au niveau logiciel, mais qu'elle permet d'éviter beaucoup plus d'erreurs. Entre autres, les fautes causées par des effets transitoires et une partie des fautes présentes sur l'en-tête des paquets ne sont pas prises en charge dans le cas de la gestion logicielle. De plus, la technique au niveau matériel se démarque par son temps de vérification nul étant donné que la vérification et l'exécution de l'application sont effectuées de façon simultanée. Pour obtenir le rapport de performance le plus élevé, ce sont le bit de parité, comme mécanisme de détection, et la reconfiguration instantanée des canaux qui sont utilisés.

La technique de prévention présentée, bien qu'elle soit développée sur le RoC, pourrait être implémentée sur d'autres réseaux sur puce multidimensionnels. Cette méthode permet de gérer autant les fautes permanentes que transitoires. Cela signifie que les circuits présentant des défauts de fabrication pourraient quand même être utilisés. D'autres circuits pourraient en tirer des bénéfices en étant plus robustes pour répondre à des applications critiques.

ABSTRACT

With the advancement in technology, embedded systems are continuously asked to support more functionalities and that brings them in becoming more and more complex. The networks on chip (NoCs) appeared in embedded systems to help managing this new complexity. A lot of NoC architectures have been proposed in the last few years, each differentiating from one another by their topology, switching technique, routing technique and so on. The Rotator on Chip (RoC), a NoC architecture developed by Polytechnique and STMicroelectronics, is one of them.

The technology scaling caused NoCs to become more vulnerable to faults. These faults can affect the system permanently, like in the case of manufacturing defects, or transiently, with magnetic interferences for instance. In order to attain a certain reliability level, fault tolerance methods must be implemented in systems. The objective is that the systems are able to achieve their functionalities even in the presence of faults. In the case of NoCs, the purpose is to make sure that no bit of the transferred packets will be corrupted. That can be done using different concepts like detection, redundancy, retransmission, correction and so on.

This thesis highlights the use of fault tolerance mechanisms in NoCs by describing the main concepts involved in NoC implementation, the faults which can occur in NoCs and the fault tolerance methods used in some specific cases.

The elaboration of a fault tolerance method adapted to the RoC characteristics is the main focus of this thesis. The proposed technique is a preventive method, rather than a corrective method. When a fault is detected on the data path, the channel reconfiguration prevents this fault from affecting other packets. Among the different implementations being compared are the layer where the preventive method is managed, the detecting mechanisms and the reconfiguration methods. Those implementations are evaluated in terms of residual packet error rate, average latency, verification time and cost in resources.

Simulations show that a method managing at the hardware level, although more expensive than one managing at the software level, is more effective. This hardware method prevents many more errors and requires no verification time. The best performing rate is obtained with parity bit as the detecting mechanism and with instant channel reconfiguration.

This preventing technique could be integrated in others multidimensional NoCs. Furthermore, it could be used to allow circuits with manufacturing defects to be used in some cases and to allow NoCs to execute critical applications.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES SIGLES ET ABRÉVIATIONS

La liste des sigles et abréviations présente, dans l'ordre alphabétique, les sigles et abréviations utilisés dans le mémoire ou la thèse ainsi que leur signification. En voici quelques exemples :

ARQ	Automatic Repeat Request
BER	Bit Error Rate
BIST	Built-in Self-test
CRC	Cyclic Redundancy Check
DUT	Design Under Test
FDAR	Fault Diagnosis And Repair
FEC	Forward Error Correction
FIFO	First In First Out
FIT	Failures In Time
Flit	Flow control digits
FPGA	Field Programmable Gate Array
GP	Generalized Petersen
HARQ	Hybride ARQ/FEC
ID	Identification
LSB	Least Significant Bit
LUT	Lookup Table
MCC	Mesh Connected Crossbars
MSB	Most Significant Bit
MSN	Manhattan Street Network
MTBF	Mean Time Between Failures
OVM	Open Verification Methodology

R^2NoC	Ring Road NoC
RPER	Residual Packet Error Rate
RoC	Rotator on Chip
RRA	Round Robin Arbiter
RTL	Register Transfer Level
SEE	Single Event Effects
SET	Single Event Transient
SEU	Single Event Upset
SHE	Single Hard Error
SPIN	Scalable Programmable Interconnection Network
VCR	Vertical Redundancy Checks
XGFT	eXtended Generalized Fat Tree

LISTE DES ANNEXES

INTRODUCTION

Les systèmes intégrés sont de plus en plus utilisés dans le but d'optimiser des applications dédiées. Nous n'avons qu'à penser aux téléphones cellulaires ou aux lecteurs MP3 qui font partie intégrante de notre vie quotidienne. Ils peuvent également avoir des applications plus critiques comme dans le domaine de la santé, de l'automobile, de l'aérospatial, etc. Avec l'avancement des technologies, ces systèmes ne cessent de gagner en complexité dans le but de remplir toujours plus de fonctionnalités. Pour augmenter les performances et répondre à toutes les fonctionnalités, il faut se tourner vers des systèmes présentant de plus en plus d'éléments de traitement.

Dans le but d'optimiser ces systèmes, deux principaux axes peuvent être analysés : les traitements de calcul et les communications entre les différents éléments. En solution au deuxième axe, différents designs ont été étudiés. Les communications point-à-point ont longtemps été utilisées, mais plus le nombre d'éléments à interconnecter augmente, plus cette solution devient impensable étant donné le nombre de liens nécessaires. Les bus ont longtemps été (et sont encore) une alternative en connectant toutes les ressources sur un bus qui gère les accès à l'aide d'un arbitre. Cette architecture a, par contre, beaucoup de limitations au niveau de la latence des paquets et de la bande passante. Une solution proposée pour éviter ces problèmes est l'utilisation des réseaux sur puce.

Les réseaux sur puce utilisent les mêmes principes que les réseaux informatiques permettant de faire communiquer plusieurs usagers ensemble. En plus d'offrir une bande passante beaucoup plus grande, ils permettent une bonne adaptation au nombre d'éléments à connecter et des communications plus rapides. Diverses architectures de réseaux sur puce ont été développées ces dernières années notamment en se basant sur les topologies de la maille et de l'anneau qui restent des paradigmes dans le monde des réseaux. Le Rotator-on-Chip (RoC) (Hadjiat et al., 2007), architecture développée en partenariat par l'École Polytechnique de Montréal et STMicroelectronics, fait partie de ces réseaux sur puce utilisant une topologie en anneau.

Problématique

Tous les réseaux sur puces ont des caractéristiques propres à eux qui les rendent plus attrayants dans certains domaines, notamment le nombre de ressources supportées, la rapidité des communications, l'énergie nécessaire, le nombre de ressources utilisées, etc. Au niveau des communications, bien que le débit soit une des métriques les plus importantes, il ne faut pas non plus oublier la fiabilité du transit des données. C'est pourquoi de plus en plus de réseaux sur puce sont développés dans une optique de tolérance aux fautes.

« […] fault tolerant systems which must continue to meet their functional and performance requirements in the presence of failures [….] » (Douglass Locke, 1994)

Avec la taille des systèmes sur puce devenant de plus en plus petite, ces systèmes sont maintenant beaucoup plus vulnérables aux fautes. Il est devenu essentiel pour rencontrer un certain niveau de fiabilité d'intégrer des méthodes de protection pour contrer l'effet des fautes dans ces systèmes. En plus qu'il est presque impossible de fabriquer des circuits de la taille d'un nanomètre sans aucun défaut, différentes sources induisant du bruit dans les réseaux sur puce mettent en péril leur fiabilité. Ces sources peuvent être internes au circuit lui-même telles les variations de tension et la diaphonie (*crosstalk*), mais elles peuvent également provenir de l'extérieur comme les interférences magnétiques et les effets des particules alpha en témoignent.

La quête de la tolérance aux fautes n'est pas nouvelle en soi dans le domaine des réseaux sur puce et plusieurs recherches ont été menées sur ce sujet. Le but ultime est qu'aucune donnée ne soit corrompue, autrement dit, qu'aucun bit lors de la transmission ne soit modifié ou, tout simplement, n'arrive pas à destination. Pour ce faire, dans la majorité des cas, l'erreur est tout d'abord détectée et, par la suite, traitée. Dans la littérature, les méthodes de détection d'erreur les plus utilisées sont le contrôle par redondance cyclique (*cyclic redundancy check*, *CRC*) (Peterson, 1960), le code de Hamming (Hamming, 1950) et le bit de parité. Une fois l'erreur détectée, il faut définir de quelle manière elle sera traitée. Dans certains cas, d'un point de vue énergétique, quand les communications se font point-à-point sur une courte distance, il est préférable d'opter pour

une technique de retransmission. Par contre, pour des connexions plus longues, il est préférable d'opter pour une technique de correction d'erreur (Bertozzi, Benini, & De Micheli, 2005). Il est également possible d'amalgamer plusieurs stratégies existantes dans le but de trouver celle qui est la plus performante pour la topologie et l'utilisation de notre réseau sur puce; c'est essentiellement l'objectif principal de ce projet. Cette composition peut être effectuée à différents niveaux, notamment en évaluant le niveau de bruit présent sur chaque connexion (Li, Vijaykrishnan, Kandemir, & Irwin, 2003) ou en associant un mode de contrôle des erreurs différent à chacun des sous-modules de notre réseau.

Objectifs

L'objectif principal de ce mémoire est de concevoir une technique de tolérance aux fautes adaptée aux caractéristiques du RoC. Plusieurs techniques de tolérance aux fautes ont déjà été développées pour les réseaux sur puce, soit pour énoncer des concepts de base, comme la détection et la correction, ou pour répondre à des besoins bien précis pour certaines architectures. Il s'agira donc d'être capable de tirer profit de toutes ces techniques en créant un amalgame pour augmenter la tolérance aux fautes du RoC qui sera évaluée en termes de nombre de fautes évitées, de latence moyenne des paquets et de ressources utilisées. Par la suite, il sera possible de comparer la méthode présentée à d'autres méthodes existantes.

Un second objectif est de comparer des techniques de tolérance aux fautes gérées au niveau logiciel avec d'autres intégrées directement au matériel. Cela sera effectué à partir des techniques de tolérance aux fautes élaborées pour le RoC.

Méthodologie

La première étape de ce travail est de définir l'architecture de réseau sur puce utilisée et de définir ses caractéristiques. Il sera important ici d'utiliser une architecture de réseau sur puce novatrice dont il est possible d'avoir accès à l'implémentation. Le RoC étant développé en collaboration avec STMicroelectronics et Polytechnique, cela permet d'avoir accès au code source de ce réseau sur puce et, de plus, il s'agit d'un nouveau design de réseau, ce qui répond

aux deux requis préalablement établis. Lors de la définition du fonctionnement et des caractéristiques du RoC, il sera possible de mettre en lumière certaines particularités qui pourront être exploitées lors du développement de la technique de tolérance aux fautes pour la rendre plus performante et efficace. Nous pouvons penser notamment à son aspect multidimensionnel. Bien entendu, parallèlement à cette étape, une étude sur l'état de l'art sera effectuée dans le but d'aiguiller le travail sur le RoC.

La deuxième étape est de définir une méthode de tolérance aux fautes utilisant les particularités du RoC qui peuvent se retrouver dans d'autres réseaux. Pour permettre de définir la technique la plus performante et efficace, plusieurs variantes devront être explorées. Les deux axes principaux seront la technique au niveau logiciel et celle intégrée au niveau matériel. Des modèles RTL (*Register Transfer Level*) des différentes techniques seront implémentés à l'intérieur du réseau choisi (RoC) pour permettre de les comparer.

L'étape finale sera de déterminer quelle technique de tolérance aux fautes s'avère être la meilleure. La performance des techniques sera évaluée en fonction du temps de vérification nécessaire, du nombre de fautes évitées, de la latence moyenne des paquets et du nombre de ressources utilisées. Tous ces résultats seront obtenus à l'aide des modèles RTL développés lors de l'étape précédente.

Contribution

Le RoC est une architecture de réseau sur puce nouvellement développée qui ne possédait pas de mécanisme pour s'assurer d'une certaine fiabilité au niveau du transfert des données. L'une des contributions les plus importantes de ce travail est de doter le RoC d'un mécanisme de tolérance aux fautes répondant à certains critères de fiabilité au niveau des données.

À un niveau plus étendu, la technique de tolérance étant développée en exploitant certaines particularités du RoC, notamment son aspect multidimensionnel, elle pourrait être intégrée à d'autres réseaux possédant les mêmes caractéristiques pour en retirer les mêmes avantages. Cette

méthode permet de gérer autant les fautes permanentes que transitoires en ne nécessitant aucun matériel de rechange. Cela signifie que les circuits présentant des défauts de fabrication pourraient quand même être utilisés et, également, que les circuits pourraient être plus robustes pour répondre à des applications critiques.

Distribution des chapitres

Ce mémoire est divisé en quatre principales sections. Le chapitre 1 débute par une revue de littérature regroupant deux principaux concepts : les réseaux sur puce et la tolérance aux fautes. En ce qui concerne les réseaux sur puce, le fonctionnement général, les topologies de base ainsi que quelques nouvelles architectures sont présentées. Pour ce qui est de la partie portant sur la tolérance aux fautes, qui est plus explicite, il est discuté de la provenance des fautes, des techniques de modélisation et des méthodes de tolérance. Le chapitre 2 présente l'architecture du RoC, son fonctionnement et ses caractéristiques, plus en détail.

Le chapitre 3 décrit l'implémentation des différentes techniques de prévention des fautes qui ont été développées, que ce soit en utilisant une gestion au niveau logiciel ou au niveau matériel. Finalement, le chapitre 4 présente les résultats obtenus dans le but de comparer les différentes variantes des méthodes.

CHAPITRE 1
REVUE DE LITTÉRATURE

Avec les systèmes sur puce grandissant et utilisant toujours plus de ressources, c'est dans le but de continuer de rencontrer les exigences en termes de performance et de consommation d'énergie que les réseaux sur puce ont été développés. La fiabilité des réseaux sur puce est continuellement reconsidérée avec les technologies de plus en plus petites qui apportent leur lot de gains mais, également, de désavantages. Malheureusement, bien que la réduction de taille permette de gagner notamment en aire, les liens deviennent très susceptibles au bruit, que ce soit par la diminution de la tension et de l'espacement entre les liens ou par l'augmentation de la fréquence d'horloge. Les sources de bruit peuvent être autant de nature interne, si nous pensons à la diaphonie, qu'externe dans le cas des interférences magnétiques, par exemple. Différentes sources de bruit qui peuvent entraîner différents types de fautes seront décrites un peu plus loin dans cette section. Pour faire face à cette réalité, des méthodes ont été développées dans le but d'éviter la propagation de ces erreurs. Les principes de base de ces méthodes sont la détection et la correction des erreurs. Il sera également fait état un peu plus loin dans cette section de plusieurs méthodes qui ont fait leurs preuves et de nouvelles méthodes qui ont été développées dans le but de toujours augmenter l'efficacité de tolérance aux fautes.

1.1 Les réseaux sur puce

Le rôle principal des réseaux sur puce est d'interconnecter plusieurs ressources sans dégrader le débit de données. Chaque réseau a des caractéristiques qui lui sont propres et qui le rend plus attrayant dans certains domaines, notamment le nombre de ressources supportées, la rapidité des communications, etc. Ces caractéristiques dépendent directement des choix architecturaux et du design du réseau. Plusieurs types d'interconnexions sont présentement utilisés pour les systèmes à plusieurs ressources, dont les bus et les réseaux sur puce. Malheureusement, les bus sont dotés d'une faible possibilité d'évolution. Plus le nombre d'éléments augmente, plus les performances diminuent de façon drastique. Les réseaux sur puce, par leur aspect évolutif, sont capables de

répondre aux carences des bus. Par contre, ils sont limités par leur aire et leur consommation (Arteris, 2005).

1.1.1 Les couches des réseaux sur puce

Les réseaux sur puce sont constitués de différentes couches : la couche physique, la couche liaison, la couche réseau, la couche transport et, finalement, la couche logicielle (Benini & Micheli, 2006).

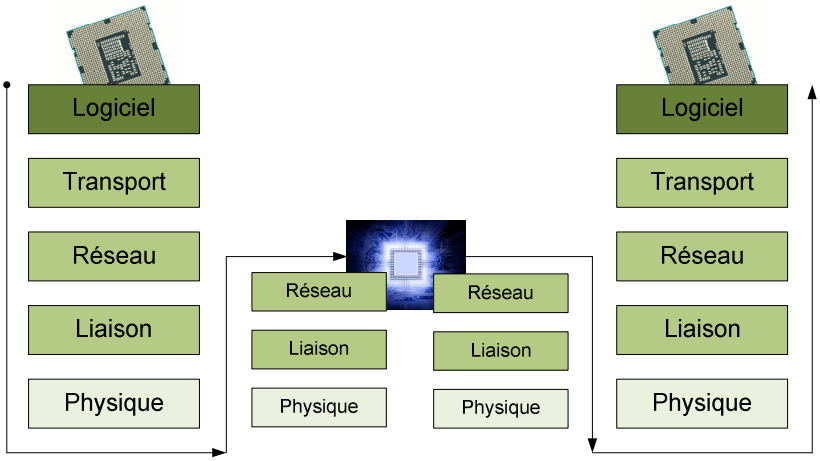


Figure 1-1: Couches des réseaux sur puce

1.1.1.1 La couche physique

La couche physique consiste en l'implémentation physique. Pour un réseau sur puce, cette implémentation se traduit par les canaux de communication par où la transmission des messages est effectuée.

1.1.1.2 La couche liaison

La couche liaison, quant à elle, va abstraire les données transmises par les canaux de communication en une liaison. Cette liaison est considérée presque sans erreur par la couche réseau. Cette couche permet d'assurer un certain niveau de fiabilité des données qui ne peut pas

être atteint avec la couche physique. Pour ce faire, cette couche transmet les données sous forme de paquet ou de *flit* (*flow control digits*), ce qui facilite la détection et la correction des erreurs.

1.1.1.3 **La couche réseau**

La couche réseau permet de gérer le routage des paquets créés par la couche liaison qui représente le chemin emprunté pour aller de la source à la destination. Également, elle permet de définir le type d'interconnexion et les différents sous-réseaux.

1.1.1.4 **La couche transport**

La couche transport est l'entité responsable de la bonne transmission des messages. Elle obtient les messages à transmettre de la couche logicielle, les fragmente en paquets plus petits si nécessaire et les transmet à la couche réseau. Lors de la réception des paquets, elle effectue les opérations contraires dans le but de reconstruire le message envoyé rendu à destination.

1.1.1.5 **La couche logicielle**

La couche logicielle ici comprend le logiciel matériel, qui correspond à une abstraction de la plateforme, et le logiciel de l'application. Le logiciel matériel peut être associé à la connexion des ressources au réseau habituellement effectuée via des interfaces nœud. Le logiciel application, quant à lui, est le logiciel s'exécutant directement sur les ressources connectées au réseau.

1.1.2 L'implémentation générale des réseaux sur puce

Chaque réseau sur puce a des caractéristiques propres en termes de latence, de débit, d'aire, d'énergie consommée, de fiabilité, etc. Ces caractéristiques sont directement reliées aux choix architecturaux notamment à la commutation, au routage et à la topologie.

1.1.2.1 **Les techniques de commutation**

La commutation dans les réseaux sur puce prend principalement deux formes : la commutation de circuits et la commutation par paquets. La commutation de circuits établit un lien unique entre le transmetteur et le receveur. Lorsque ce lien est réservé, il envoie le message entier par ce même chemin. Si, lors de l'établissement du lien, il y a conflit avec un chemin déjà réservé, l'envoi du message est retardé jusqu'à la libération du chemin en conflit. C'est la technique de commutation qui est utilisée dans le cas du SoCBUS (Wiklund & Dake, 2003), car elle demande un minimum de mémoire et permet d'éviter les interblocages. La commutation par paquets, quant à elle, ne fait aucune réservation de liens et tous les paquets fragmentés du message sont indépendants selon leur chemin. Étant donné qu'il n'y a pas de réservation de chemin, si plusieurs paquets veulent obtenir le même lien, un seul pourra l'avoir et les autres devront attendre.

1.1.2.2 **Les techniques de routage**

Le routage des paquets est la façon dont le chemin de la source à la destination est défini. Dans le cas de la commutation par paquets, il y a trois principales techniques : stockage et retransmission (*store and forward*), *virtual cut through* et trou de ver (*wormhole*). Le stockage et retransmission est la technique la plus simple qui consiste à ce que le paquet soit envoyé seulement quand il est assuré que le récepteur a suffisamment d'espace mémoire pour le recevoir. Le *virtual cut through* est similaire à la technique précédente dû au fait que la destination doit être capable de recevoir tout le paquet, mais il se distingue par le fait que ce sont des *flits* qui sont envoyés en cascade et non pas des paquets entiers. Cela permet donc des transferts de données plus rapides. Finalement, le trou de ver se distingue du *virtual cut through* par le fait qu'il nécessite de l'espace mémoire pour seulement un *flit* et non pas un paquet entier. Donc, si l'espace mémoire disponible n'est pas égal à la largeur d'un paquet entier, les *flits* d'un paquet peuvent être dispersés dans plusieurs routeurs. C'est cette technique qui obtient les meilleures performances en termes de mémoire nécessaire et de latence, mais elle est plus susceptible aux interblocages.

1.1.2.3 **Les composants de base**

Les quatre composants de base à toute architecture de réseaux sur puce sont : l'interface réseau, les nœuds, les interconnexions et les canaux de communication. L'interface réseau sert en quelque sorte d'adaptateur et permet aux ressources externes de se connecter et d'utiliser le réseau sur puce. Les nœuds sont des éléments de calcul, mais aussi de mémoire, qui correspondent aux points d'entrée et de sortie des messages dans le réseau. Les interconnexions servent à acheminer les messages de la source à la destination et peuvent être intégrées au nœud ou non. Les canaux de communication relient les interconnexions ou les nœuds, selon le cas, entre eux.

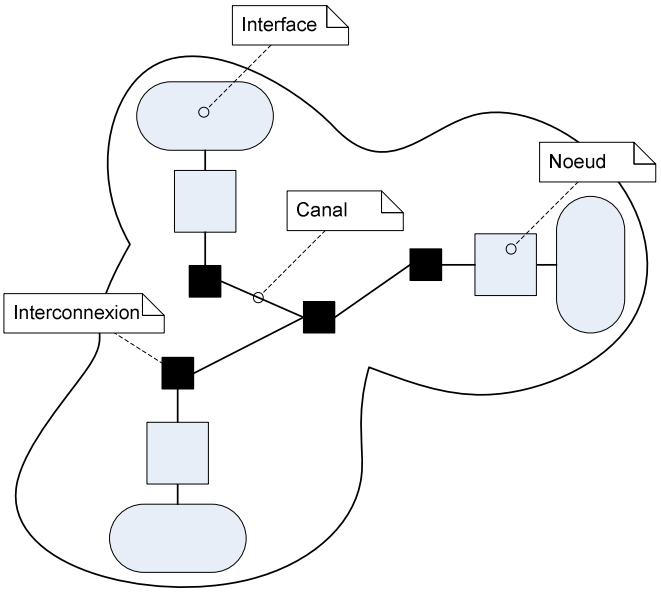


Figure 1-2: Éléments d'un réseau sur puce

Dans le cas où les interconnexions sont intégrées aux nœuds et où ces derniers ont des liaisons point-à-point entre eux, ces réseaux sont considérés comme des réseaux directs. Dans le cas où les interconnexions sont détachées, mais ont des connexions entre elles, ce sont des réseaux indirects. Des solutions hybrides peuvent aussi exister.

1.1.3 Les topologies des réseaux sur puce

Les topologies des réseaux sur puce sont très similaires à celles des réseaux informatiques classiques. La topologie définit comment les ressources sont reliées entre elles. Chaque disposition obtient des caractéristiques particulières, que ce soit en cas de panne ou pour sa technique de routage. Parmi les topologies de réseaux sur puce, nous comptons notamment la maille, l'arbre et l'anneau, mais il en existe bien plus ayant chacune quelques variantes.

1.1.3.1 La topologie en maille

La topologie en maille est l'une des plus simples et des plus étudiées. Elle consiste à ce que tous les nœuds soient connectés avec leurs voisins, ce qui permet une certaine fiabilité car, en cas de panne d'un lien ou d'un nœud, d'autres chemins sont possibles. La forme la plus simple est une topologie de maille en deux dimensions où chaque nœud est associé à un commutateur, comme présentée à la Figure 1-3 . Par contre, d'autres architectures sont possibles. Notamment, une alternative à la maille de base est le *Mesh Connected Crossbars* (MCC) (Tavakkol, Moraveji, & Sarbazi-Azad, 2008) qui connecte les nœuds via des barres croisées (*crossbars*) sur les diagonales du réseau et point-à-point aux bordures. Cette topologie a pour avantage de diminuer la densité de canaux. Cette architecture utilise une technique de routage adaptative qui permet d'acheminer les paquets à travers le réseau en empruntant le chemin minimal.

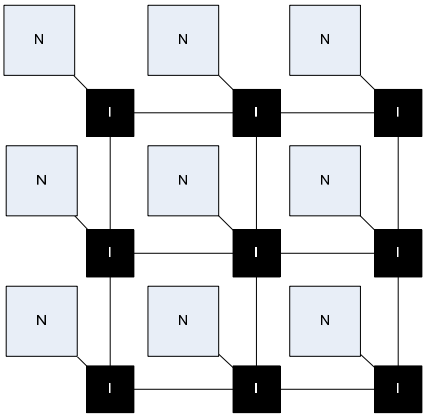


Figure 1-3: Topologie maille

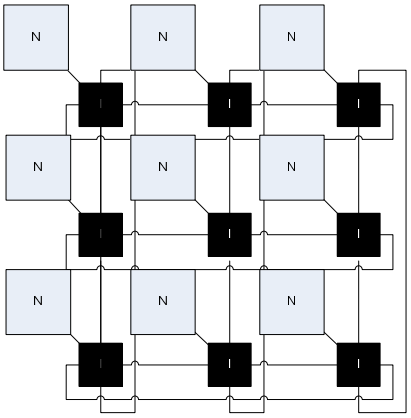


Figure 1-4: Topologie tore

Dérivé de la topologie en maille, il existe également le tore (*torus*) qui est très similaire à la maille, mais qui complète en quelque sorte les liens du réseau en reliant le dernier nœud au premier sur chaque axe, comme il est présenté à la Figure 1-4. Cette topologie est notamment utilisée dans l'architecture du *Manhattan Street Network* (MSN) (Oommen & Harle, 2005). Le MSN a la particularité que le sens des liens alterne d'une colonne ou d'une ligne à l'autre (verticalement et horizontalement). Comme dans le modèle de base, chaque nœud est associé à une interconnexion qui est en fait une barre croisée. Dans cette architecture, le trafic sur le réseau a priorité sur le trafic spécifique aux nœuds. Également, une autre topologie similaire à celle de la maille et du tore est l'*hypercube* où les nœuds représentent les sommets du cube et les liens, les arêtes.

1.1.3.2 **La topologie en arbre**

La topologie en arbre fait appel à la hiérarchie et à la relation parents-enfants. Le nœud au sommet peut être connecté à plusieurs nœuds à un niveau inférieur, qui peuvent eux-mêmes être connectés à d'autres nœuds de niveaux inférieurs, et ainsi de suite. Un des risques de cette topologie est que si l'un des nœuds parents tombe en panne, ce sont tous ses enfants qui sont mis hors service. Une topologie dérivée est celle de l'arbre élargi (*fat tree*) qui est en réalité une topologie en arbre où les nœuds peuvent seulement être au dernier niveau (feuilles). Par contre, dans cette topologie, les interconnexions aux niveaux supérieurs peuvent avoir plusieurs connexions entre elles, comme présentées à la Figure 1-5. Parmi les architectures d'arbre élargi, nous comptons notamment le SPIN (Scalable Programmable Interconnection Network) (Adriahantenaina, Charlery, Greiner, Mortiez, & Zeferino, 2003) et le XGFT (*eXtended Generalized Fat Tree*) (Kariniemi & Nurmi, 2005) qui utilisent des liens bidirectionnels.

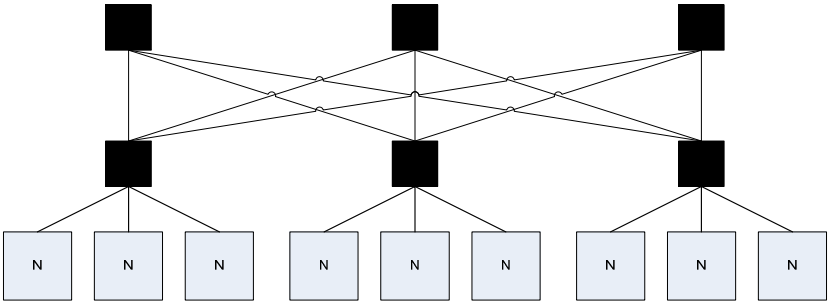


Figure 1-5: Topologie d'arbre élargi

1.1.3.3 **La topologie en anneau**

La topologie en anneau relie les nœuds d'un réseau entre eux afin de former une boucle comme à la Figure 1-6. Chaque nœud entre la source et la destination sert alors d'interconnexion. Parmi les architectures de réseau sur puce utilisant la topologie anneau, notons notamment le Proteo, le ftNoC et le RoC. L'implémentation du RoC sera décrite plus en détail dans le chapitre suivant. Le Proteo (Siguenza-Tortosa & Nurmi, 2002) peut être utilisé pour des systèmes hétérogènes divisés en clusters. Le réseau est composé d'un anneau bidirectionnel avec des sous-réseaux de topologie bus ou en étoile. Les ressources sont connectées au réseau via des nœuds qui sont reliés entre eux par des liens. Le réseau est paramétrable à plusieurs niveaux, notamment la profondeur des tampons, le nombre de canaux, leur dimension, la largeur des données, la segmentation des paquets, etc. Dans le but d'économiser de la puissance, il est possible d'éteindre des sous-réseaux en bloquant le nœud qui y est attaché. Comme mécanisme de tolérance aux fautes, les nœuds fautifs sont déconnectés de façon permanente.

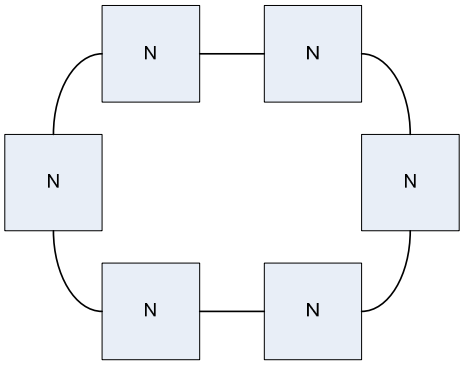


Figure 1-6: Topologie anneau

Pour ce qui est du ftNoC (Liguo, Huimin, & Jungang, 2008), il est composé de deux groupes d'anneaux dont l'un est utilisé dans le but de remplacer le premier s'il s'avère défaillant. Chaque groupe d'anneaux est composé d'un anneau de contrôle et d'un de données. Il y a trois types de nœuds qui sont utilisés dans ce réseau : les nœuds de contrôle, les nœuds de données et les nœuds d'interface qui servent à assurer l'interface entre le réseau principal et les sous-réseaux. L'architecture proposée permet d'éviter la congestion, les interblocages, la famine et assure une utilisation efficace de la bande passante. Chaque nœud a un mécanisme de détection des erreurs

sur les liens de données et les liens de contrôle dans le réseau. S'il y a une erreur sur les données, l'information est propagée à tous les nœuds du réseau par les liens de contrôle. Les données ne sont plus envoyées sur ce lien et l'autre anneau de données est utilisé. S'il y a une erreur sur le contrôle, il y a un changement direct qui est effectué sur le second anneau de contrôle.

Une autre architecture qui peut s'apparenter en quelque sorte à celle de l'anneau est le GP(2m,1) (*Generalized Petersen*) (Liu, Jungang, & Du, 2008). Cette architecture présente deux anneaux qui sont interconnectés entre eux. Cette topologie a l'avantage d'être régulière. Tout comme pour la topologie en anneau, les nœuds n'ont pas besoin de l'information complète du réseau, mais seulement celle du nœud courant et de celui de destination. Lorsqu'un nœud reçoit un paquet, il vérifie s'il doit l'envoyer au nœud suivant ou au nœud adjacent. Les paquets sont acheminés par le chemin le plus court. Une autre architecture présentant plusieurs anneaux reliés entre eux est le *Ring Road NoC* (R^2NoC) (Samuelsson & Kumar, 2004). Ici, les anneaux sont bidirectionnels et deux types de nœuds sont utilisés, ceux connectant les ressources au réseau et ceux reliant les anneaux entre eux.

1.1.3.4 **La topologie en étoile**

La topologie en étoile consiste à relier tous les nœuds par une connexion centrale. L'avantage de cette configuration est que si un nœud tombe en panne, le reste du réseau peut continuer à fonctionner normalement. Par contre, si la connexion centrale tombe en panne, c'est le réseau complet qui est affecté. Une variante de cette topologie est NOVA (Martinez Vallina, Jachimiec, & Saniie, 2007), une architecture en étoile « augmentée », qui est présentée à la Figure 1-7. En fait, ce sont des tuiles de huit nœuds reliés par un commutateur Banyan (*Banyan switch*) qui sont disposées selon une topologie en étoile utilisant le même type d'interconnexion. Les communications locales d'une tuile peuvent s'effectuer directement entre deux nœuds voisins ou en utilisant l'interconnexion centrale. Par contre, les transferts de données entre les tuiles peuvent seulement être faits en utilisant la connexion centrale.

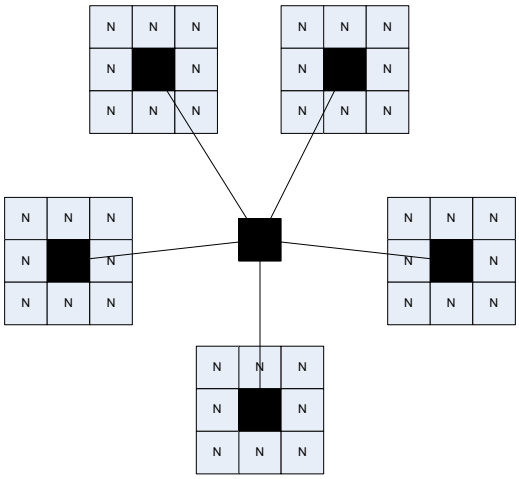


Figure 1-7: Topologie étoile augmentée (NOVA)

1.1.3.5 **Autres topologies**

Certaines architectures ont été développées en utilisant des caractéristiques de différentes topologies. Le Spidergon (Concer, Iamundo, & Bononi, 2009) en est un exemple. Les nœuds sont disposés sur un anneau bidirectionnel et sont reliés entre eux par une barre croisée comme dans une structure en étoile. Si la destination est proche, les communications se feront par l'anneau et, dans le cas contraire, par l'interconnexion centrale. Comme plusieurs types de réseau sur puce, le Spidergon utilise comme technique de commutation le trou de ver pour l'économie qu'elle permet en termes de mémoire.

1.2 Les fautes dans les réseaux sur puce

Les avancées technologiques poussent toujours à obtenir les meilleures performances avec des composants toujours plus petits. Les interconnexions des systèmes sur puce ne cessent de diminuer de taille également et sont maintenant rendues à l'échelle du nanomètre. Cette diminution de taille amène les interconnexions à devenir de plus en plus sensibles et vulnérables aux fautes (Shivakumar, Kistler, Keckler, Burger, & Alvisi, 2002). Ces fautes peuvent avoir un effet permanent aussi bien que transitoire. Elles peuvent exister à la création même du système, un défaut de fabrication par exemple, ou elles peuvent être causées par des perturbations extérieures lors de son utilisation.

1.2.1 Causes et classification des fautes

Principalement, il existe deux grandes catégories de fautes : celles qui présentent un effet permanent et celles qui ne durent que de façon transitoire. Ces fautes peuvent avoir lieu autant au niveau des éléments de mémoire que des circuits logiques. Un routeur peut faire une erreur de routage sans pour autant corrompre la donnée, ce qui résulte à une erreur lors de la transmission du paquet dans un cas comme dans l'autre.

Parmi les fautes permanentes, nous pouvons noter deux types, soit statique et dynamique. Les fautes statiques se produisent généralement lors de la fabrication du réseau, qui nécessite des techniques de plus en plus minutieuses, et leur effet est irréversible excepté par des modifications physiques du réseau. Les fautes dynamiques, étant de nature permanente également, ont lieu de façon aléatoire durant l'exécution, contrairement aux fautes statiques. Elles peuvent être allouées aux effets de vieillissement, à l'électromigration, etc.

Les fautes transitoires sont, quant à elles, de nature non permanente. Tout comme les fautes dynamiques, elles apparaissent de façon aléatoire sur le réseau, mais se résorbent d'elles-mêmes. Elles sont souvent associées au bruit présent sur le réseau. Le fait que la tension dans le réseau soit diminuée, cela peut rendre les connexions plus susceptibles à la diaphonie (*crosstalk*), au

bruit de couplage et aux erreurs logiques (*soft errors*) (Baumann, 2000). Elles peuvent également être dues à des radiations, des sauts et des interférences électromagnétiques. Les erreurs logiques peuvent être divisées en deux catégories : erreurs sur les liens et erreurs intrarouteurs.

Les *single event effects* (SEE) sont des événements perturbant le comportement d'un circuit dû à une grande charge de particules qui peuvent résulter par la corruption de données. Leur effet peut être aussi bien permanent que transitoire.

Parmi ces SEE, nous en comptons deux principaux : les *single event upsets* (SEU) et les *single event transients* (SET) (Bastos, 2010). Dans le cas des SEU, ce sont des particules cosmiques qui entrent en contact avec le circuit et qui créent un changement d'état dans un élément logique tel une mémoire ou un registre.

Les SET sont également dus à des particules cosmiques et peuvent être propagés à travers du circuit. Ils sont généralement représentés sous forme de pulsions (*glitches*). Cela peut résulter en une mauvaise donnée transmise si l'événement atteint un loquet (*latch*) ou une bascule (*flip flop*). Habituellement, les effets des SEU et des SET sont transitoires.

Il peut également exister des SEE qui ont des effets permanents tels les *single event hard errors* (SHE) qui consistent en un bombardement d'énergie sur certains bits, ce qui peut souvent mener à une valeur forcée (*stuck-at).*

1.2.2 Caractérisation des fautes

Les fautes pouvant être créées par différentes sources, de la même façon, il est possible de les injecter de différentes façons dans les circuits. Parmi les méthodes d'injection physiques, nous comptons notamment la variation de la tension d'alimentation, de l'horloge (Ozev, Sorin, & Yilmaz, 2007) et de la température ainsi que le flash de lumière, la source de particules ou le laser (Monnet, 2007). Il est possible d'injecter les fautes à différents niveaux, soit physique,

logique et fonctionnel. Nous nous concentrerons sur la modélisation au niveau logique, comme la majorité des études analysées. Quelques avantages de la modélisation au niveau logique sont que la représentation est assez minutieuse (portes logiques) pour permettre de simuler des effets physiques sans avoir à procéder à des bombardements de particules, par exemple, et que le modèle est indépendant des technologies utilisées.

1.1.2.2 Métriques utilisées

Une des métriques utilisées dans la modélisation des fautes est le taux d'erreur sur les bits (*bit error rate*, BER). Le BER correspond au ratio de bits en erreur et est défini comme étant le nombre de bits en erreur divisé par le nombre total de bits transmis. L'équation correspondante est présentée ci-dessous. Le nombre de bits en erreur est le nombre de bits reçus qui ont été altérés lors de leur transfert sur le canal de communication (Khalili & Salamatian, 2005).

$$BER = \frac{Nombre\ de\ bits\ en\ erreur}{Nombre\ total\ de\ bits\ transmis} \qquad (1)$$

D'autres métriques permettant d'évaluer les fautes influant sur un système sont le MTBF et le FIT. L'intervalle moyen entre les défaillances (*mean time between failures*, MTBF) correspond au temps moyen entre deux fautes (équation 2). Le nombre de défaillances dans le temps (*failures in time*, FIT) représente quant à lui le nombre de fautes qui peuvent avoir lieu pendant 10^{-9} heures d'utilisation. La relation entre MTBF et FIT est présentée à l'équation 3.

$$MTBF = \frac{Temps\ d'exécution\ total}{Nombre\ de\ fautes} \qquad (2)$$

$$MTBF = \frac{1\ 000\ 000\ 000}{FIT} \qquad (3)$$

Une métrique permettant d'évaluer la fiabilité de différentes méthodes de tolérance aux fautes consiste à les comparer en utilisant le taux d'erreur sur les paquets résiduels (*residual packet*

error rate, RPER) (Bertozzi et al., 2005) qui correspond à la probabilité qu'un paquet erroné soit encore présent après le contrôle d'erreur. Ce taux peut également être défini à d'autres niveaux, notamment en fonction des messages ou des bits. Également, plusieurs recherches utilisent la latence moyenne des paquets, l'aire nécessaire ainsi que l'énergie consommée pour définir la performance des techniques.

1.2.2.1 **Modélisation des fautes**

Comme il a été mentionné précédemment, les fautes peuvent être dues à différentes causes autant internes, tels les bruits de puissance et la diaphonie, qu'externes comme l'effet de particules alpha ou l'électromigration. De ce fait, la probabilité des fautes est différente pour chaque circuit et dépend de plusieurs facteurs, notamment les facteurs environnementaux, les paramètres de design du circuit et l'usage du circuit à proprement dit (Hass, 1999). En ce qui concerne les effets que le design peut avoir sur la probabilité des erreurs, nous savons que le ratio d'erreurs augmente avec la complexité du circuit, la densité, la diminution du voltage, la diminution des latences et la diminution de la capacitance des cellules (Tezzaron, 2004).

Concernant les éléments de mémoire, une étude (Tezzaron, 2004) démontre que, pour différentes technologies, le ratio d'erreur de type erreur logique peut aller de 4^{-7} à 5^{-14} erreur/bit/heure. Ce qui donne en moyenne, 1000 à 5000 FIT/Mbit. De plus, environ 2% de ces erreurs peuvent avoir un effet permanent sur le réseau. Ces données sont obtenues au niveau du sol et les ratios d'erreurs ont tendance à augmenter avec l'altitude. Il est fait mention (MoSys, 2002) que le ratio peut augmenter de cinq (5) fois à 2600 pieds et de dix (10) fois à une altitude de 5280 pieds.

Le Tableau 1-1 présente les probabilités pour différents types de fautes qui peuvent affecter les réseaux sur puce, tirées du livre « Networks on Chip » de Benini et De Micheli (Benini & Micheli, 2006).

Tableau 1-1: Probabilités de différents types de fautes sur un réseau sur puce

Type de faute	Modèle de la faute	Valeur	Unité
Bruit gaussien sur un canal	Transitoire	10^{-20}	BER
Effet particule alpha sur un élément de mémoire	Erreur logique	10^{-9}	SER
Effet particule alpha sur un élément logique	Transitoire	10^{-10}	BER
Électromigration	Valeur forcée	1	MTBF (FITs)
Changement de seuil	Valeur forcée	1	MTBF (FITs)
Corrosion d'un connecteur	Valeur forcée	10	MTBF (FITs)
Joint de soudure	Valeur forcée	10	MTBF (FITs)
Défaillance de tension	Arrêt	10^4	MTBF (FITs)
Défaillance logicielle	Arrêt	10^4	MTBF (FITs)

Au niveau de la modélisation des fautes, (Young Hoon, Taek-Jun, & Draper, 2010) utilisent l'injection de faute statistique (*statistical fault injection*, SFI) (Mukherjee, Emer, & Reinhardt, 2005) qui injecte des changements de bits aléatoires autant en termes de temps que de localisation. Cette technique est appliquée à chaque bit de chaque lien dans le routeur avec des probabilités indépendantes. Ils évaluent leur méthode à différents niveaux de taux d'erreur. Ils évaluent leur technique pour différents taux d'erreur allant de 10^{-6} à 10^{-3} (erreur/fil (*wire*)/cycle), ce qui est très haut et qui représente un cas extrême pour des fautes transitoires. Cela permet de mettre le réseau sous stress. D'autres testent leurs méthodes sur un éventail plus large utilisant un BER allant de 10^{-15} à 10^{-2} (Li et al., 2003).

Dans certains cas (Qiaoyan & Ampadu, 2011), les fautes sont modélisées selon des scénarios de fautes transitoires seulement et de fautes permanentes seulement. D'autres (Ababei & Katti, 2009) évaluent avec des cas de défaillances simples et de défaillances multiples simultanées. Il peut aussi arriver que des erreurs logiques variées soient injectées de façon aléatoire dans les routeurs et les liens inter-routeurs (Park, Nicopoulos, Kim, Vijaykrishnan, & Das, 2006).

1.3 Méthodes de gestion des fautes pour les réseaux sur puce

Pour éliminer les fautes pouvant subvenir sur le réseau ou, du moins, pour en diminuer les effets, plusieurs méthodes ont été développées, que ce soit en travaillant sur la structure physique des circuits (en ajoutant des boucliers ou en modifiant la charge des substrats) ou sur la structure logique (en modifiant le logiciel et/ou le matériel).

Pour la prise en charge des fautes au niveau logique, les principes de base sont la détection et la correction. Plusieurs méthodes ont été étudiées autant dans un cas que dans l'autre.

Les deux principales formes de correction des erreurs sont la retransmission et l'utilisation de codes correcteurs. Dans le cas de la retransmission, un code de détection des erreurs est ajouté au paquet transmis. Si le récepteur détecte une erreur à l'aide du code de détection, il commande alors une retransmission au transmetteur. Dans le cas d'une correction d'erreur à proprement dit, un code correcteur est ajouté au paquet ce qui permet au receveur de corriger lui-même l'erreur si tel est le cas. Les erreurs transitoires peuvent être gérées en utilisant un code de contrôle d'erreur dans la couche des liens de données. La retransmission est une solution économique au niveau de l'énergie quand peu de bruit est présent dans le composant. Sinon, on peut faire appel à des codes de correction, mais ceux-ci nécessitent plus de ressources et d'énergie par le fait même. Dans d'autres cas, des solutions hybrides utilisant la retransmission et la correction sont utilisées.

La détection peut avoir lieu à différents niveaux : soit au niveau « réseau » en utilisant une vérification bout en bout, qui correspond à vérifier la donnée rendue à destination, ou soit au niveau « liens » en effectuant une détection connexion à connexion, qui vérifie la donnée à tous ses sauts à l'intérieur du réseau. La technique bout en bout amène des délais pour la détection des fautes, encore plus lorsqu'un paquet est perdu dans le réseau et qu'il faut le détecter. L'approche au niveau des liens entre les routeurs (interconnexions) présente beaucoup de bénéfices (Dally, Dennison, Harris, Kan, & Xanthopoulos, 1994). En effet, la retransmission au niveau des liens demande des tampons de retransmission moins larges. Elle garantit la communication bout en bout, tout en ne nécessitant pas de confirmation de réception particulière.

1.3.1 Méthodes de redondance de base pour la tolérance aux fautes

Certaines techniques de détection et de correction des fautes faisant appel à la redondance ne nécessitent plus de faire leurs preuves et sont utilisées dans beaucoup de méthodes plus poussées qui ont été développées ces dernières années. Parmi ces techniques bien établies, notons notamment les suivantes:

1.3.1.1 **Bit de parité**

Le bit de parité est la méthode de détection la plus simple. Il consiste simplement à ajouter un bit à la fin du message pour vérifier la parité (Tanner, 1984). Il y a deux méthodes pour coder la parité : parité paire et parité impaire. Dans le cas de la parité paire, le nombre de 1 présent dans le message doit être pair, alors que c'est le contraire pour la parité impaire. La figure ci-dessous présente la méthode de calcul d'un bit de parité, il s'agit simplement d'appliquer des OU exclusif sur tous les bits entre eux.

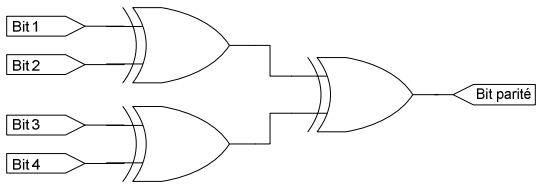


Figure 1-8: Circuit du calcul du bit de parité

Lors de l'envoi, la parité est calculée et ajoutée à la fin du paquet et, lors de la réception, celle-ci est recalculée et comparée avec celle ajoutée lors de l'envoi.

Il est spécifié que le bit de parité permet la détection d'erreur simple. En fait, il permet de détecter tout nombre d'erreurs impair. Le fait d'ajouter un seul bit pour la détection des erreurs donne une méthode nécessitant très peu de ressources et d'énergie (Palframan, Nam Sung, & Lipasti, 2011).

1.3.1.2 **Somme de contrôle**

La somme de contrôle, communément appelée *checksum*, correspond à la somme de tous les bits du message qui est ajoutée à la fin du message dans le but de comparaison. Cette technique permet de détecter des erreurs multiples sur le message sans toutefois pouvoir clairement les indiquer. De plus, plus la somme de contrôle est effectuée sur un large message, plus elle perd de son efficacité, car les agencements de bits pouvant donner la même somme sont plus fréquents. La technique du contrôle de parité est une variante de la somme de contrôle.

1.3.1.3 **Contrôle par redondance cyclique (CRC)**

Le CRC est une méthode qui consiste également à ajouter un code à la suite du paquet dans le but de procéder à la validation des données. Cette technique a été développée par W. Wesley Peterson (Peterson, 1960). C'est sûrement la technique de code cyclique la plus connue et la plus utilisée encore aujourd'hui. Le principe de base résulte d'une division polynomiale entre le message et le polynôme générateur dont le reste est ajouté à la fin du paquet comme code de validation (Sobolewski, 2003). Un exemple de calcul de CRC est illustré en annexe. Tout comme les autres méthodes mentionnées précédemment, l'encodage du message se fait par le transmetteur et le receveur le compare avec celui des données reçues. En réalité, lorsque le message est reçu par le receveur, le calcul du CRC devrait donner 0 si aucun bit n'a été altéré, car le reste a déjà été ajouté au paquet. Cette technique reste très facilement réalisable au niveau matériel.

Un des avantages du CRC par rapport au contrôle de parité est la détection de paquets d'erreurs (*burst errors*), c'est-à-dire plusieurs erreurs qui peuvent alors lieu en séquence sur un même message. Le contrôle de parité, pair pour être plus précis, est un cas spécial du CRC où le polynôme générateur est : $X + 1$.

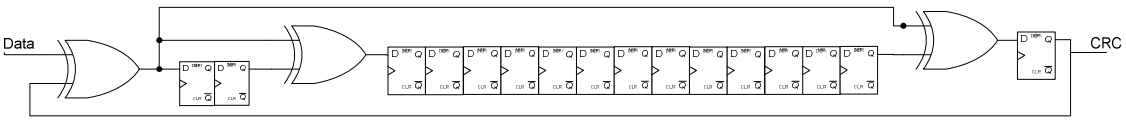


Figure 1-9: Circuit du calcul du CRC-16

1.3.1.4 **Code de répétition**

Le principe de base du code de répétition est d'envoyer plusieurs fois les mêmes bits dans le but de les comparer. Dans ce cas, le nombre d'envois le plus efficace est trois (*triple modular redundancy*). De cette façon, les trois paquets sont comparés et, si un d'entre eux est erroné, il sera détecté.

1.3.1.5 **Codes correcteurs**

Il existe également des codes correcteurs qui, en plus de détecter les erreurs, permettent de les corriger en les localisant. Il existe notamment le code de Hamming (Hamming, 1950) qui permet la correction d'erreur simple seulement. Également, les codes BCH et Reed-Solomon, qui permettent des corrections multiples, mais qui nécessitent substantiellement plus de ressources.

1.3.2 « Nouvelles » méthodes pour la tolérance aux fautes

Les principes de base concernant la prise en charge des fautes permanentes et transitoires sont que les fautes permanentes peuvent être évitées en reconfigurant le réseau pour que les zones touchées ne soient plus empruntées et que les fautes transitoires peuvent être détectées par un code de détection. Par la suite, les paquets touchés peuvent être enlevés, pour éviter que l'application ne prenne une donnée corrompue en considération, ou corrigés. Comme mentionné à la section *1.3 Méthodes de gestion des fautes pour les réseaux sur puce*, le terme correction comprend ici la retransmission autant que l'utilisation de codes correcteurs.

1.3.2.1 **Méthodes de prévention**

Suivant ces principes, en ce qui concerne les fautes permanentes, certaines techniques au niveau réseau ont été étudiées. Parmi celles-ci, nous comptons notamment l'envoi de plusieurs paquets similaires (inondation) et la possibilité de différents chemins à emprunter. Ceci peut s'avérer efficace pour des erreurs sévères. Les algorithmes d'inondation permettent une excellente tolérance aux fautes car, si un chemin existe, le message est assuré d'arriver à destination. Par

contre, dans ce type d'algorithmes, beaucoup de paquets sont transmis sans être nécessaires. Alors, une diminution de transmission des paquets inutiles résulte en une augmentation du débit et en une meilleure utilisation des interconnexions (Pirretti et al., 2004). Au niveau physique, des liens et des composants de rechange (Shamshiri & Kwang-Ting, 2009) ou des transmissions divisées (*split transmissions*) (Lehtonen, Liljeberg, & Plosila, 2007) peuvent être utilisés pour un faible nombre d'erreurs permanentes. En termes de performance et d'énergie, l'ajout de fils supplémentaires est plus efficace que les transmissions divisées. Par contre, les transmissions divisées nécessitent un moins gros coût en surface.

1.3.2.2 **Techniques de détection**

Pour la gestion des fautes autant permanentes que transitoires, certains utilisent une détection bout en bout avec retransmission pour les fautes transitoires et une routine déterministe pour éviter les liens où des erreurs permanentes sont présentes (Bertozzi et al., 2005). D'autres, allant dans un tout autre sens, ont évalué l'attrait d'une méthode de routage stochastique, qui est en fait le contraire d'une routine déterministe, sur une plateforme reconfigurable dynamiquement (Nunez-Yanez, Edwards, & Coppola, 2008). L'avantage de cette routine dans le cas de la reconfiguration dynamique est que les tables de routage n'ont pas besoin d'être tenues à jour.

Les liens de données étant l'un des composants les plus vulnérables, plusieurs ont développé des techniques dans le but de les protéger contre les principales sources de fautes dans un réseau. Des pistes qui ont été obtenues sont de garder l'aire et la tension nécessaires à un minimum. Dans l'article (Park et al., 2006), les auteurs se penchent sur les fautes présentes sur les liens de données mais, également, sur les fautes intrarouteur qui se retrouvent à l'intérieur des composants des routeurs. Les erreurs sur le chemin de données intrarouteur peuvent être détectées comme des erreurs sur les liens à la validation au prochain routeur. Il est expliqué que le principe de retransmission nécessite des tampons pour garder les paquets en mémoire, mais permet de gérer des erreurs multiples de cette façon, comparativement aux méthodes de correction. La retransmission bout en bout peut être beaucoup affectée par des erreurs touchant l'en-tête, car le paquet peut être acheminé à une mauvaise destination. Aussi, si la source est corrompue, la

demande de retransmission ne peut pas être transmise, ce qui est évité dans le cas d'une retransmission connexion à connexion.

Une autre étude (Ejlali, Al-Hashimi, Rosinger, & Miremadi, 2007) évalue les techniques *Automatic Repeat Request* (ARQ), *Forward Error Control* (FEC) et *Hybride ARQ/FEC* (HARQ) selon des critères de fiabilité, de consommation d'énergie et de performance au niveau connexion à connexion. Le ARQ correspond à la retransmission du *flit*, qui est en fait la plus petite décomposition d'un paquet, quand une erreur est détectée jusqu'à l'obtention d'une transmission sans faute. Le FEC permet de corriger une erreur sur un bit sans demande de retransmission. Le HARQ utilise les deux techniques donc, il corrige les erreurs simples et demande une retransmission pour les erreurs multiples. Pour les mêmes critères de fiabilité, le ARQ consomme moins d'énergie que le FEC, mais le HARQ obtient les meilleurs résultats. En ce qui concerne le niveau de bruit, lorsqu'il est bas, le ARQ est préférable, mais le HARQ le devient à son tour lorsque le niveau de bruit augmente. Pour ce qui est des contraintes de temps, le FEC peut s'avérer favorable.

Plus récemment, l'article (Young Hoon et al., 2010) présente un flot de contrôle tolérant aux fautes pour gérer les erreurs logiques dans les réseaux sur puce. Le flot permet de couvrir les erreurs au niveau des liens en utilisant des retransmissions. Ceci est effectué grâce au principe de fragmentation dynamique des paquets qui retransmet la partie du *flit* en erreur par la partie non contaminée de la communication. La méthode de détection utilisée est un CRC-8. Quand un *flit* est reçu, la validation des données est faite grâce au CRC et, en même temps, la donnée est emmagasinée dans un canal virtuel. De cette façon, le routeur peut détacher le *flit* en faute des autres *flits* qu'il garde en mémoire. Il est à noter que le CRC est effectué en parallèle, ce qui n'augmente pas le chemin critique. Par contre, cette technique limite le débit, car les *flits* sont retenus plus longtemps qu'à l'ordinaire. Cette méthode présente une couverture d'erreur de 97 % pour les éléments du chemin de données avec une augmentation de 13,7 % en termes d'aire.

1.3.2.3 **Méthodes adaptatives**

Diverses méthodes ont été explorées pour permettre de gérer les fautes permanentes et transitoires au niveau du chemin de données et des couches physiques également. Certaines utilisent une unité de contrôle de codage configurable pour définir le nombre de liens supplémentaires nécessaires en fonction du bruit présent (Qiaoyan & Ampadu, 2010). Cela permet une meilleure gestion de l'énergie. Ces liens sont utilisés soient en redondance, dans le cas de fautes transitoires, ou en rechange, dans le cas de fautes permanentes. Cette méthode permet de diminuer la latence des paquets et l'énergie consommée comparativement à d'autres. L'utilisation d'une unité de contrôle de codage configurable pour les fautes transitoires au niveau des liens de données peut également être jumelée à un algorithme de réorganisation des paquets et une transmission divisée pour les erreurs permanentes au niveau physique (Qiaoyan & Ampadu, 2011). Certains, utilisant le même principe de contrôle des erreurs adaptatif, utilisent un lien de données supplémentaire (Li et al., 2003). Ce lien permet d'évaluer la quantité de bruit sur le réseau et, à partir du résultat obtenu, le schéma pour le contrôle des erreurs est adapté. De cette façon, ils s'assurent de répondre aux exigences de fiabilité tout en consommant le moins d'énergie possible.

Une autre approche (Kologeski, Concatto, Carro, & Kastensmidt, 2011) recueille des informations sur l'emplacement des défauts pour mettre en action les composants de tolérance aux fautes quand ceux-ci sont nécessaires. Avec l'emplacement des fautes, il est possible de désactiver un composant en faute et d'activer une méthode de tolérance aux fautes. Cette approche étant adaptative, elle nécessite moins d'énergie, car elle est seulement activée en cas de besoin et elle utilise la division des données au besoin. Quand une faute est détectée, une routine adaptative essaie d'éviter ce lien, sinon la division des données est utilisée en envoyant les paquets en deux parties en évitant les parties endommagées. Cette technique est basée sur les tests de fabrication pour définir les zones en faute et elle représente une augmentation d'environ 28 % de l'aire.

1.3.2.4 **Comparaison de la prise en charge entre les niveaux réseau et physique**

Les différences de performance entre les techniques au niveau réseau et celles au niveau lien ont été comparées dans certains cas. Notamment, (Murali et al., 2005) présente l'évaluation de trois différents mécanismes de contrôle des erreurs (retransmission, correction, hybride) autant au niveau du réseau que des liens de données. Les principaux critères étudiés sont l'efficacité énergétique, l'efficacité de la protection contre les erreurs et l'impact sur la performance du réseau. Le bit de parité et le CRC sont les codes utilisés pour évaluer les différents schémas de détection dans le cas de la retransmission. Pour les détections bout en bout, ils sont ajoutés au paquet, tandis qu'ils sont ajoutés au *flit* pour les détections connexion à connexion. Le mécanisme hybride correspond plus spécifiquement à une correction d'erreur simple et à une retransmission des erreurs multiples.

Les résultats obtenus en ce qui concerne la consommation d'énergie démontrent que la détection avec bit de parité consomme plus d'énergie, car elle a une moins grande capacité de détection des erreurs que la technique hybride et que celle utilisant le code CRC.

Pour ce qui est de la performance des méthodes, elle est évaluée pour quatre différentes méthodes : bout en bout, connexion à connexion au niveau des paquets, connexion à connexion au niveau des *flits* et hybride. La méthode bout en bout s'avère désavantageuse au niveau de la latence. La détection connexion à connexion au niveau des paquets engendre une latence un peu plus élevée qu'au niveau des *flits* et c'est la méthode hybride qui obtient le meilleur résultat. Pour les quatre mêmes méthodes, au niveau de la consommation d'énergie, c'est encore le bout en bout qui obtient les moins bons résultats et l'hybride les meilleurs. Il est démontré que dans le cas de longs liens ou de beaucoup de sauts, le technique bout en bout est à privilégier. Dans le cas contraire, mieux vaut utiliser une approche connexion à connexion.

Le Tableau 1-2 présente les avantages et inconvénients des principales méthodes de correction et de vérification des fautes.

Tableau 1-2: Comparaison des méthodes de correction et de vérification des fautes

Principes	Avantages	Inconvénients
Retransmission	• Préférable pour des communications sur de courtes distances. • Moins énergivore.	• Nécessite des mémoires tampon pour conserver les paquets envoyés. • Délais de retransmission. • Nécessite une confirmation de réception.
Codes correcteurs	• Aucun délai de retransmission. • Préférable pour des communications sur de longues distances.	• Complexité de l'implémentation. • Nécessite plus de ressources. • Plus énergivore.
Vérification bout en bout	• À privilégier avec des longs liens ou beaucoup de nœuds.	• Délais pour la détection des fautes. • Fautes touchant l'en-tête peuvent avoir de grandes répercussions. • Augmentation de la latence.
Vérification connexion à connexion	• Retransmission demande des tampons moins larges. • Garantit également les communications bout en bout.	• Devient complexe avec beaucoup de nœuds.

1.3.2.5 **Vérification au niveau logiciel**

Certaines techniques, quant à elles, se développent plus du côté de la vérification logicielle même dans le cas des composants matériels. Ceci permet de diminuer la complexité et le nombre de ressources utilisées par le matériel. Dans un certain cas, une architecture de réseau sur puce en arbre, *eXtended Generalized Fat Tree* (XGFT), avec un nouveau système de diagnostique de faute et réparation (*fault-diagnosis-and-repair*, FDAR) est utilisé à l'aide du bit de parité comme méthode de détection (Kariniemi & Nurmi, 2005). La méthode FDAR permet une reconfiguration du routeur du nœud où la faute est présente ainsi que de la table de routage du réseau pour s'assurer que la faute détectée soit évitée et n'affecte pas d'autres paquets. La méthode FDAR est implémentée en logiciel.

Dans les XGFTs, étant donné que les données ont un seul chemin possible en descente, un diagnostic de fautes est présent pour permettre de localiser ces dernières. Le XGFT fournit également plusieurs redondances de ressources, dont plusieurs interconnexions et des liens bidirectionnels, ce qui permet d'améliorer la tolérance aux fautes. Une fois qu'une faute est détectée, il est possible par une routine déterministe d'acheminer les paquets dans les zones sans faute.

Pour la détection des fautes, les auteurs utilisent un bit de parité par octet et des contrôles par redondance verticale (*vertical redundancy checks*, VCR) qui peuvent permettre la correction d'un bit. La vérification est effectuée au niveau des interconnexions. Si une erreur est détectée dans l'en-tête du paquet, le paquet est enlevé et un paquet vide est transmis. Si un paquet peut être routé correctement, la faute n'est pas considérée assez sérieuse. Un BIST (*built-in self-test*) est effectué sur les canaux tout de suite après la réinitialisation du réseau. Le BIST échoue si le nombre d'erreurs est supérieur à un nombre prédéterminé.

Les auteurs utilisent différents modes, déterminés dans l'en-tête du paquet, pour router les paquets dans le réseau (un mode normal, un mode pour le routage déterministe, un pour le routage déterministe avec diagnostic et un mode pour la reconfiguration).

1.3.2.6 **Éléments de traitement**

D'autres auteurs se sont penchés sur les fautes affectant les éléments de traitement. Dans un contexte de réseau, la tolérance aux fautes correspond à la capacité de maintenir l'habilité de transmettre les messages en présence de composants en faute. Dans le cas où c'est un élément de traitement qui est en faute, sa fonction doit alors être déplacée sur un ensemble d'éléments de traitement différent. Cela doit être fait rapidement et avec le moins de déplacements possible. Pour Ababei et Katti, le déplacement et le partage de la nouvelle configuration sont effectués grâce à un gestionnaire général qui est implémenté sur une tuile du réseau (Ababei & Katti, 2009). Les deux étapes sont de trouver de nouveaux éléments et, par la suite, procéder au déplacement.

CHAPITRE 2
ROTATOR ON CHIP

Le réseau sur puce qui est étudié et utilisé dans le but de développer une nouvelle technique de prévention des fautes est le *Rotator-on-Chip* (RoC) (Hadjiat et al., 2007). Cette nouvelle architecture a été développée par l'École Polytechnique de Montréal en collaboration avec STMicroelectronics. Ceci a donc permis d'avoir accès à l'implémentation de base du réseau, ce qui a été d'une très grande utilité.

Comme mentionné précédemment, le RoC présente une nouvelle architecture de réseau sur puce basée sur la topologie de réseau en anneau. Le réseau est complètement évolutif en ce qui concerne sa configuration (nombre de nœuds et de canaux, nombre de FIFOs d'entrée et largeur du chemin de données). En plus d'être un réseau multidimensionnel, ce qui veut dire qu'il comporte plusieurs canaux de communication, il permet un flot de communication bidirectionnel. D'autres caractéristiques du RoC permettent de l'ajuster au besoin de chaque application, notamment les classes de service qui permettent d'associer une priorité différente à chaque FIFO d'entrée du RoC dans le but de mieux gérer le trafic. Également, une optimisation du chemin de données est possible, permettant de transférer les paquets en plusieurs segments dans le but de diminuer le nombre de ressources matérielles nécessaires.

2.1 Vue d'ensemble

Comme tous les réseaux, le but du RoC est de permettre la communication entre différentes ressources. Ici, les ressources connectées au RoC peuvent être de différentes natures : un processeur, une mémoire, un composant matériel, etc. Les ressources sont connectées aux nœuds à l'aide d'adaptateurs. De cette façon, différents protocoles de communication peuvent être utilisés. Par la suite, chaque ressource est connectée au réseau à l'aide d'une interface nœud. C'est cette interface qui permet l'envoi et la réception des paquets sur le réseau. Elle est constituée principalement des FIFOs d'entrée et de sortie et du *add/drop controller*. Les données

transitant sur le réseau sont gérées, quant à elles, par les banques qui sont constituées de tampons. Ce sont ces deux composants (interface nœud et banque) qui constituent le nœud à proprement dit. La Figure 2-1 présente ces composants sur une configuration du RoC à quatre (4) nœuds et deux (2) canaux.

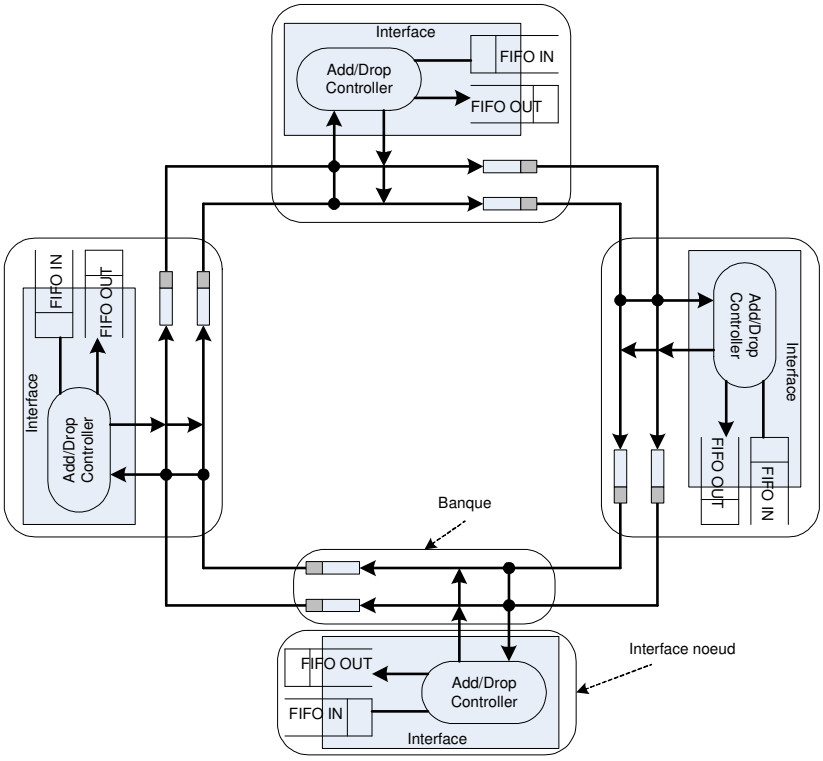


Figure 2-1: Architecture du RoC avec 4 nœuds et 2 canaux

La caractéristique du RoC permettant d'avoir plusieurs canaux de communication en parallèle offre la possibilité de transférer plusieurs paquets simultanément. De plus, le fait que le nombre de canaux de communication soit extensible permet de modifier la bande passante.

2.2 Architecture du RoC

2.2.1 Interface Nœud

Les ressources envoient des messages sur le réseau par l'interface nœud. L'interface crée le paquet d'envoi avec les informations nécessaires et l'injecte sur le RoC. Parmi ces informations, nous comptons notamment l'ID de la source, l'ID de la destination, le mode du paquet, la priorité, la taille des données et, bien entendu, les données. L'interface nœud comporte notamment les processus de vérification des adresses de destination, d'extraction des paquets, d'injection des paquets et de sélection des canaux. À chaque cycle d'horloge, un paquet par nœud peut être extrait et ajouté.

2.2.2 Banque

Chaque interface est connectée à une banque qui contient des tampons. Chaque tampon constitue une partie d'un anneau qui correspond à un canal. Alors, un canal est constitué de *N* tampons. Un tampon peut contenir un paquet et le transfert de l'information se fait selon une rotation horaire. Un paquet est transmis de la source à la destination en passant d'une banque à l'autre sans être interrompu.

2.2.3 Nœud

Chaque tampon peut contenir un et un seul paquet. Les paquets sont transférés de la source à la destination en passant de tampon à tampon jusqu'à temps d'avoir rejoint cette dernière sur le canal de communication déterminé. Pour assurer l'ordonnancement des messages, ceux qui sont déjà en transit sur le RoC ont priorité sur les nouveaux messages injectés. De cette façon, les messages envoyés sur le réseau ne peuvent être arrêtés ou retardés lors de leur transit. Ce principe peut être comparé à un carrefour giratoire. Lorsqu'une voiture est engagée à l'intérieur, elle a priorité sur les autres voitures qui veulent y entrer. Lorsque la voiture quitte le carrefour, une nouvelle place est libérée pour une voiture qui voulait entrer. Pour ce faire, chaque tampon est

équipé d'un signal de contrôle permettant de définir son statut, soit occupé ou libre. Ce signal sert également à l'extraction des paquets.

La Figure 2-2 présente l'architecture complète d'un nœud. Les messages sont envoyés et reçus de la ressource par les FIFOs d'entrée et de sortie. Comme mentionné précédemment, il est possible de configurer le RoC avec plusieurs FIFOs d'entrée et de leur assigner des priorités différentes pour avoir une meilleure gestion du trafic et favoriser des communications plus critiques. Le *add-drop controller*, qui est vraiment la partie de contrôle de l'interface nœud, est présenté sur la partie centrale et la banque où les paquets transitent est représentée sur la droite.

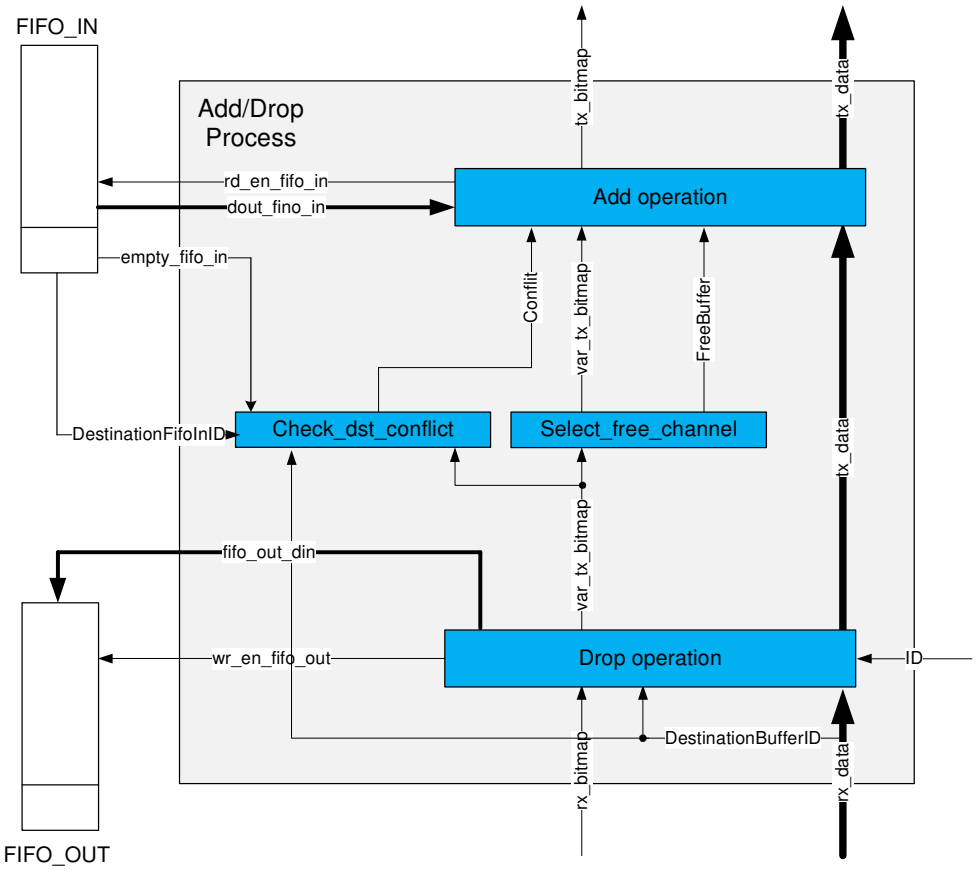


Figure 2-2: Implémentation d'un nœud du RoC

Les deux opérations fondamentales pour un nœud sont l'injection et l'extraction de paquets.

L'extraction se fait lorsqu'un paquet arrive à sa destination. Il est alors écrit sur la FIFO de sortie du nœud de destination et son tampon est libéré.

L'injection se fait lorsqu'un paquet doit être envoyé à une destination. Dans le cas de l'injection, une condition importante doit être remplie :

C.1) qu'il y ait un tampon de libre et qu'il n'y ait pas déjà, dans un autre tampon, un paquet pour cette destination en transit sur le RoC.

Notez ici que C.1 assure l'ordonnancement des paquets.

En résumé, l'ordre des opérations effectuées à chaque cycle dans le nœud est le suivant :

1) **Extraction d'un paquet** : l'adresse de destination du paquet est comparée à celle du nœud. Si les deux concordent, le paquet est extrait du tampon et est placé dans la FIFO de sortie. Pour savoir quels canaux contiennent des paquets utiles, un *bitmap* présentant l'état des canaux (libre ou occupé) est transmis d'un nœud à l'autre.
2) **Mise à jour du statut des tampons** : si une extraction a eu lieu à l'étape précédente, le tampon correspondant est libéré.
3) **Vérification des conflits de destinations** : la condition C.1 est vérifiée.
4) **Sélection d'un canal** : mise à jour des tampons ayant le statut libre.
5) **Injection d'un paquet** : si la condition C.1 est vérifiée, le paquet de la FIFO d'entrée est ajouté au tampon et son statut est maintenant occupé.

Le temps de latence pour chaque paquet à l'intérieur du RoC dépend de la distance qu'il a à parcourir. La latence correspond donc à $L + 2$ où L est le nombre de nœuds à parcourir. Donc, si un nœud envoie à son voisin, cela nécessitera trois (3) cycles : un (1) pour l'injection, un (1) pour le transit et un (1) pour l'extraction.

2.3 Optimisations

2.3.1 Flot de communication

Une communication de topologie anneau, en termes de latence, est excellente lorsque les communications sont dans le sens directionnel, mais très mauvaise dans le cas inverse. Dans le but d'éviter ce problème, un flot de communication bidirectionnel a été instancié à l'intérieur du RoC. Alors, lorsqu'un paquet est introduit dans le RoC, le calcul du chemin le plus court est effectué pour décider de la direction du paquet. Également, l'extraction des paquets se fait selon un principe de tourniquet entre les FIFOs de sortie des deux directions. Chaque nœud est donc constitué de deux noyaux ayant leur banque dans deux sens inverses. Cela est représenté sur la Figure 2-3. La direction des paquets à injecter est définie par le module *select direction* qui calcule le chemin le plus court. Les paquets sont extraits par le module *round robin arbiter* (RRA) qui sélectionne les paquets des deux directions selon un principe de tourniquet.

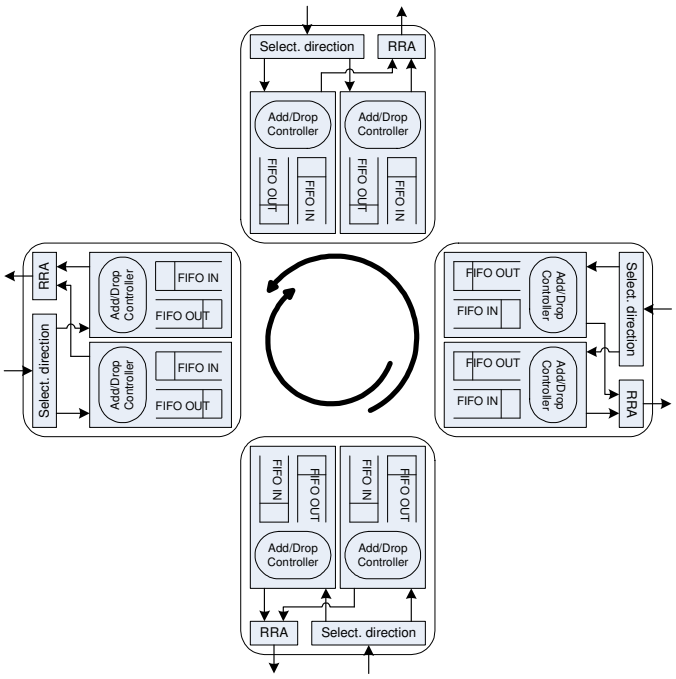


Figure 2-3: Flot bidirectionnel pour un RoC à 4 nœuds

Cette optimisation permet de diminuer la latence moyenne par un facteur deux (2). Avec cette modification, la plus grande distance à parcourir dans le RoC n'est plus *N* (*N* représentant le nombre de nœuds), mais bien *N*/2.

2.3.2 Optimisation du chemin de données

L'implémentation initiale du RoC effectue le transfert de paquets entiers entre les nœuds. Une optimisation a été apportée au RoC permettant de partitionner les paquets en différents segments pour diminuer la largeur du chemin de données et, par le fait même, le coût en aire. Dans ce cas, lorsque le premier segment est envoyé, les autres suivent sans interruption. Bien que cette caractéristique permette de diminuer le coût en ressources, elle diminue également les performances du réseau. Il faut donc bien configurer cette caractéristique selon nos besoins.

Considérons que le paquet est segmenté en K parties. Idéalement, K devrait être un diviseur de la largeur du paquet pour s'assurer d'obtenir des segments égaux. La segmentation des paquets permet de diminuer les tampons nécessaires à l'intérieur des nœuds. La largeur des tampons est maintenant égale à S/K, où S représente la largeur d'un paquet. Ceci permet donc de diminuer le coût en aire.

Par contre, le fait que les segments se suivent à travers les différents nœuds augmente la latence des paquets. Lorsqu'une opération d'injection ou d'extraction se produit, seulement un segment est injecté ou extrait à chaque cycle. Cela a pour effet d'augmenter la latence normale d'un paquet de K cycles.

L'article (Hadjiat et al., 2007) présente des résultats de simulations abondant dans le même sens. Pour un RoC bidirectionnel comportant 32 nœuds et 8 canaux (4 dans chaque direction), les auteurs obtiennent un coût en aire correspondant à 58 798 LUTs et 25 023 bascules pour un RoC n'utilisant pas la segmentation. Pour un RoC ayant les mêmes caractéristiques et utilisant la segmentation en 2 segments, ils obtiennent un total de 43 247 LUTs et 19 919 bascules. Cela représente une diminution du coût en aire de plus de 20%. Par contre, au niveau de la charge maximale supportée, un RoC bidirectionnel sans segmentation s'approche de 90% alors qu'un RoC avec segmentation en deux parties sature aux alentours de 60%. La charge maximale correspond au nombre total de paquets acheminés par rapport au transfert maximal théorique.

CHAPITRE 3
MÉTHODES DE TOLÉRANCE AUX FAUTES PROPOSÉES

3.1 Vue d'ensemble

Ayant étudié différentes techniques utilisées dans le domaine de la tolérance aux fautes, notre but est d'élaborer une méthode permettant d'éviter au moins 80% des erreurs résultant des fautes sur le chemin de données. Cela doit pouvoir se réaliser en minimisant le délai des paquets et avec une implémentation ne nécessitant pas plus que le double des ressources utilisées dans l'implémentation du RoC de base. Celle-ci est développée pour un réseau multidimensionnel et bidirectionnel. L'architecture utilisée dans ce cas-ci est le RoC qui a été défini au chapitre précédent. Deux approches permettant d'effectuer la prévention à des niveaux différents sont analysées, soit au niveau logiciel et au niveau matériel. Ici, le but des techniques proposées n'est pas de corriger les erreurs mais bien de réduire le nombre de paquets affectés par les fautes au minimum. Cette section explique les détails d'implémentation de ces différentes techniques.

3.2 Mécanismes de détection des fautes

Deux systèmes de détection d'erreur sont étudiés, soit le contrôle de parité et le contrôle par redondance cyclique (CRC).

Dans le cas du **contrôle de parité**, il s'agit d'un des mécanismes les plus simples qui peuvent être utilisés. Il s'agit simplement d'ajouter un bit, appelé bit de parité (0 ou 1), à la fin du mot. C'est la parité paire, décrite à la section *1.3.1.1 Bit de parité*, qui est utilisée tout au long de cette étude. Un de principaux avantages de cette technique est sa simplicité, autant au niveau matériel que logique. Par contre, le taux d'efficacité de cette méthode peut laisser à désirer lorsque plusieurs bits sont en erreur sur le même paquet.

Dans ce travail, c'est le CRC-16 qui sera utilisé. Il a été démontré que les CRC utilisant 16 (Koopman & Chakravarty, 2004), 24 et 32 bits s'avèrent plus performants en termes de distance

de Hamming que les protocoles précédents. Les paquets du RoC ayant une largeur de 96 bits, l'utilisation du CRC-16 semble plus adéquate. Le polynôme générateur correspondant est : $X^{16} + X^{15} + X^2 + 1$ (Castagnoli, Ganz, & Graber, 1990). Le CRC-16 ajoute un code de validation de 16 bits au message de base. Il permet de détecter les erreurs simples, doubles et de nombre impair. Également, il détecte les séquences d'erreurs de 16 bits ou moins et la majorité des erreurs dans le cas de séquences plus grandes. Le fonctionnement du contrôle de redondance cyclique est décrit plus en détail à la section *1.3.1.3 Contrôle par redondance cyclique (*CRC).

3.3 Technique de prévention des fautes au niveau logiciel

La technique de prévention des fautes au niveau logiciel est en fait une routine permettant de détecter les fautes et de prévenir la corruption des paquets transitant sur le réseau qui est entièrement gérée par les ressources (processeurs) qui sont connectées au RoC. Pour ce faire, on procède en deux principales étapes; tout d'abord, la vérification de tous les segments du RoC et, par la suite, selon les résultats obtenus, la reconfiguration du chemin de données du réseau.

La **vérification** des segments se fait à l'aide d'envois de paquets sur le réseau et de la comparaison des paquets reçus avec ceux envoyés.

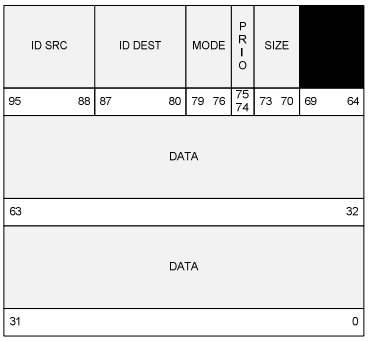
La **reconfiguration** du chemin de données consiste à désactiver les canaux où des segments ont été détectés comme étant en faute.

3.3.1 Protocole de communication

Pour permettre à la couche logicielle de commander une reconfiguration au niveau matériel, un nouveau protocole de communication a dû être implémenté. De façon générale, il y a trois sortes de paquets correspondant à trois modes différents (Figures 3-1 à 3-4). Ces paquets sont envoyés en trois envois de 32 bits. Par contre, les paquets de type « reconfiguration », comme présentés à la Figure 3-4, nécessitent seulement deux envois de 32 bits.

- **Mode application** (00):

  Les paquets « application », sont les paquets transitant normalement sur le RoC avec un en-tête de 32 bits et 64 bits de données. L'en-tête contient les champs source, destination, mode, priorité et largeur des données. Ce sont les mêmes champs d'en-tête qui sont utilisés pour tous les types de paquets.



ID SRC : identifiant de la source

ID DEST : identifiant de la destination

MODE : mode du paquet (ici 0x0)

PRIO : priorité du paquet

SIZE : taille des données (octets)

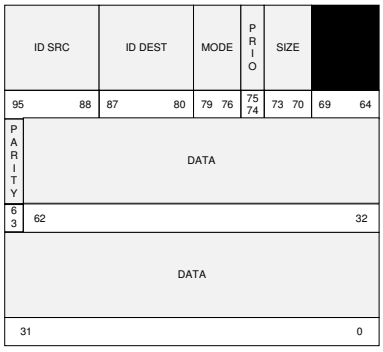
* Les cases noires sont « inutilisées » et mises à 0.

Figure 3-1: Format d'un paquet "application "

- **Mode test** (01):

  Les paquets « test » sont les paquets utilisés pour détecter les canaux fautifs.

  Dans le cas du « **test parité** », les paquets sont composés d'un en-tête de 32 bits, ainsi que de 63 bits de données et du bit de parité calculé sur la donnée.



MODE : 0x1

PRIO : 0

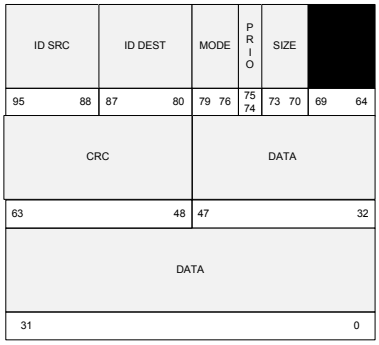
SIZE : 8 octets

PARITY : Bit de parité calculé sur la donnée

* Les cases noires sont « inutilisées » et mises à 0.

Figure 3-2: Format d'un paquet "test parité"

En ce qui a trait au « **test CRC** », les paquets sont composés d'un en-tête de 32 bits, ainsi que de 48 bits de données générés aléatoirement et du code CRC-16 qui correspond à la donnée.



MODE : 0x1

PRIO : 0

SIZE : 8 octets

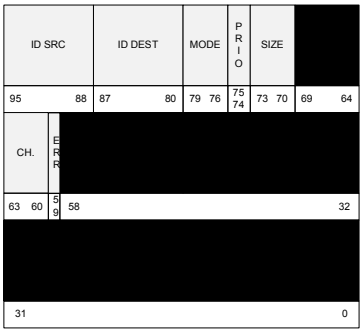
CRC : CRC calculé sur la donnée

* Les cases noires sont « inutilisées » et mises à 0.

Figure 3-3: Format d'un paquet "test CRC"

- **Mode reconfiguration** (11):

  Les paquets « reconfiguration » sont utilisés pour reconfigurer les canaux du RoC. Ils contiennent le même en-tête de 32 bits, le canal à reconfigurer et le statut à assigner à ce dernier.



ID DEST : définit la direction (i+1 ou i-1)

MODE : 0x3

PRIO : 0

SIZE : 4 octets

CH : identifiant du canal

ERR : état du canal (0 OK, 1 fautif)

* Les cases noires sont « inutilisées » et mises à 0.

Figure 3-4: Format d'un paquet "reconfiguration"

3.3.2 Modifications apportées à l'architecture du RoC

Pour que cette technique de prévention des fautes soit réalisable, quelques modifications à l'architecture du RoC ont dû être apportées et elles sont présentées dans la section suivante. En effet, les modifications apportées au module *node* pour permettre la prise en charge de la technique de prévention sont d'abord présentées. Puis, ces modifications amènent la création de modules supplémentaires dont le *bitmap_error_update* ou la modification d'autres modules dont le *select_direction* qui seront décrits plus en détail de façon subséquente.

3.3.2.1 Modifications au module *node*

Comme il a été expliqué précédemment, il y a deux principales parties à la technique de prévention: la vérification et la reconfiguration. Pour que la reconfiguration soit possible, on doit être en mesure d'activer et de désactiver les différents canaux du RoC. Pour ce faire, un paquet de mode reconfiguration comprenant l'ID du canal, la direction ainsi que l'état est envoyé. Ceci nécessite d'apporter certaines modifications au module *node*, comme présentées en hachuré sur la Figure 3-5.

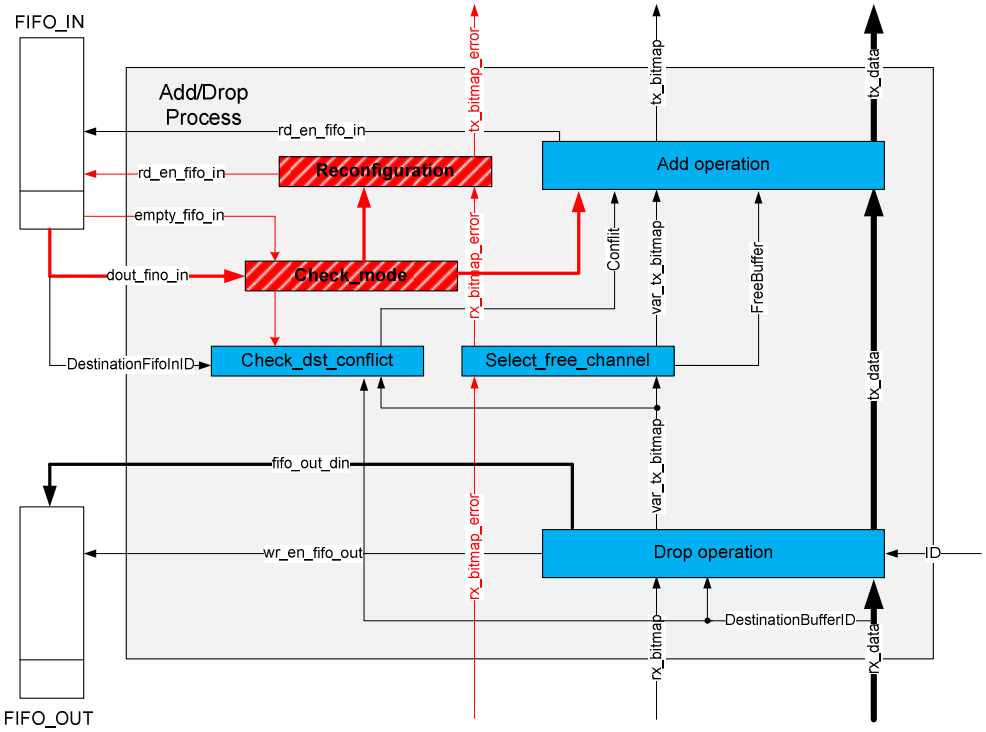


Figure 3-5 : Module *node* avec prise en charge de la reconfiguration des canaux

Il y a principalement trois grandes modifications :

1) L'ajout du *bitmap_error* : On remarque l'ajout d'un *bitmap_error* répertoriant l'état des canaux (activé ou désactivé).

2) La vérification du mode et la reconfiguration : Ici présentée avec « Check_mode ». Si le mode correspond à « application » ou « test », le processus se poursuit comme avant en vérifiant les conflits de destination et la disponibilité d'un canal. Par contre, si le mode correspond à « reconfiguration », l'ID du canal présent dans le paquet est utilisé pour mettre à jour le *bitmap_error* avec le nouvel état du canal. Il est à noter que les paquets de reconfiguration sont de priorité 0, i.e. la priorité la plus élevée.

3) La vérification du *bitmap_error* avant la sélection d'un canal : En plus de la contrainte qu'un tampon soit libre au nœud précédent, on doit s'assurer que le canal sur lequel se trouve ce tampon n'est pas considéré en faute.

3.3.2.2 Ajout du module *bitmap_error_update*

Pour que le changement du *bitmap_error* soit envoyé à tous les nœuds en même temps, un nouveau composant a été ajouté. Le Tableau 3-1 décrit l'interface du composant supposant une architecture à *N* nœuds avec *C* canaux.

Tableau 3-1: Description de l'interface du module *bitmap_error_update*

Port	Entrée/Sortie	Description
rx_bitmap_error[0..(NC)-1]	Entrée	Vecteur regroupant les *bitmap_error* provenant de tous les nœuds.
tx_bitmap_error[0..C-1]	Sortie	*bitmap_error* mis à jour et transmis à tous les nœuds.

3.3.2.3 **Modifications au module *select_direction***

Lors de leur entrée dans le RoC, la destination des paquets est évaluée par le composant *select_direction* dans le but de décider la direction à emprunter (horaire ou antihoraire). L'algorithme de base calculait la direction en fonction du chemin le plus court sans prendre en considération l'état des canaux de la direction, car ceux-ci ne pouvaient être désactivés. Par contre, avec l'ajout du module *bitmap_error* et la vérification des canaux, il est maintenant possible que tous les canaux d'une même direction soient déclarés comme étant en faute et, par le fait même, que la direction entière soit désactivée. Dans ce cas, on doit s'assurer de la poursuite du traitement de l'application et que tous les paquets soient acheminés par la direction qui demeure en service. Il est attendu que cela entraînera une augmentation considérable au niveau du délai moyen des paquets. Quelques modifications ont donc été apportées au module *select_direction* pour que cela soit possible, ce qui résulte en l'algorithme suivant :

1) **Vérification de la disponibilité des deux directions** : si tous les canaux du sens horaire sont désactivés, envoi du paquet dans le sens antihoraire. Dans le cas contraire, si tous les canaux du sens antihoraire sont désactivés, envoi du paquet dans le sens horaire.
2) **Évaluation du chemin le plus court** : évaluation du nombre de nœuds à parcourir entre la source et la destination par rapport à la moitié du nombre total de nœuds du RoC. Le paquet est envoyé sur le chemin le plus court.

3.3.3 Technique de vérification

Le but de cette technique de prévention des fautes est, tout d'abord, de détecter les fautes présentes sur le réseau et, par la suite, de reconfigurer le réseau dans le but d'éviter la propagation des erreurs. Dans notre cas, ce ne sont pas les données en tant que telles qui sont vérifiées mais bien les paquets de tests dédiés à cette tâche. Ceci permet de ne pas augmenter le chemin critique des données de l'application avec la vérification, car ce sont seulement les paquets de test qui subissent le processus avant l'exécution de l'application.. Cela est réalisé au niveau logiciel à l'aide d'une routine de vérification qui s'exerce avant le début de l'application. Chaque ressource connectée au RoC exécute une tâche de test permettant de vérifier l'intégrité du chemin de données du RoC.

Pour réaliser cette technique, deux types de ressources sont définies: maître et esclave. Il y a une seule ressource de type maître (habituellement le nœud 0) et c'est elle qui débute la routine. Toutes les autres sont de type esclave. La routine de test permet de vérifier les canaux, segment par segment, en adoptant une approche à relais. De façon générale, l'algorithme de la routine, considérant une architecture à N nœuds et à C canaux ayant comme ressource maître le nœud 0, se définit comme suit :

Tableau 3-2: Algorithme général de la technique de prévention au niveau logiciel

#	Étape
1	Le nœud 0 active seulement le premier canal dans la direction horaire à l'aide de paquets de reconfiguration.
2	Le nœud 0 envoie un paquet de mode test au nœud 1.
3	Le nœud 1 reçoit le paquet et vérifie la validité des données.
4	Si les données sont erronées ou si le délai est expiré, l'identifiant du canal sera gardé en mémoire.
5	Le nœud envoie alors un paquet de test au nœud 2 et ainsi de suite jusqu'à ce que le nœud N-1 envoie un paquet de test au nœud 0.
6	Par la suite, le nœud 0 active seulement le second canal en sens horaire et les envoie au nœud N-1, et ainsi de suite.
7	Ceci s'exécute pour les C canaux et est repris pour les C canaux dans le sens antihoraire également.
8	Quand tous les segments de tous les canaux ont été vérifiés, le nœud 0 effectue la reconfiguration complète des canaux à l'aide des données recueillies en désactivant les canaux détectés comme étant en faute.

La Figure 3-6 représente l'envoi des messages de test pour une architecture à quatre nœuds et deux canaux par direction.

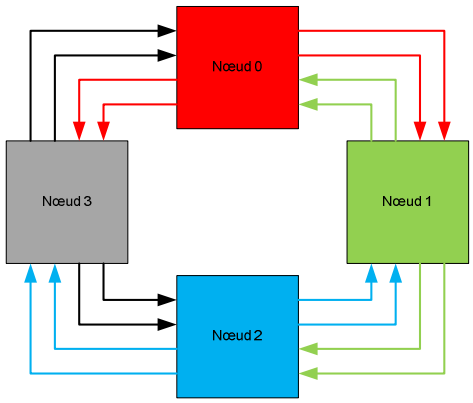


Figure 3-6: Flot de tests

3.3.3.1 **Vérification simple**

La vérification simple consiste à vérifier une seule fois chaque segment du réseau. L'algorithme est exactement celui décrit précédemment et le flot de tests correspond également à la Figure 3-6. La vérification des canaux est faite à l'aide de paquets de format « test » comme ceux présentés à la section *3.3.1 Protocole de communication*.

Considérant qu'un envoi correspond à un cycle et qu'un paquet prend sept (7) cycles pour transiter d'une ressource à une ressource voisine, nous obtenons l'équation (4) pour calculer le nombre de cycles nécessaires à la vérification complète du réseau où N et C correspondent respectivement au nombre de nœuds et au nombre de canaux du réseau. Par exemple, pour un réseau comportant 8 nœuds et 3 canaux, nous obtenons un total de 732 cycles nécessaires à la vérification et la reconfiguration.

$$T_{Vérif\_simple} = 2[2C + C(2C + 14N)] + 4C \qquad (4)$$

3.3.3.2 **Vérification double**

La vérification double consiste en tout point à la vérification simple excepté que les segments du RoC sont vérifiés deux fois plutôt qu'une. Le nombre de cycles nécessaires à la vérification du réseau est présenté par l'équation (5). Donc, pour un réseau comportant 8 nœuds et 3 canaux, nous obtenons un total de 1404 cycles nécessaires.

$$T_{Vérif\_double} = 2[2C + C(2C + 28N)] + 4C \qquad (5)$$

Il y a deux variations de la vérification double qui sont étudiées :

Code CRC-16 : La technique de vérification à l'aide d'un code de contrôle s'avérant plus efficace que celle avec un simple bit de parité, c'est cette technique qui a été retenue pour la vérification double. Les paquets de test sont les mêmes que ceux présentés à la Figure 3-3 et utilisés précédemment.

Séquences 0 et 1 : Les segments sont vérifiés successivement avec des paquets comportant un en-tête de 32 bits et 64 bits de données. Pour la première vérification, les données de test envoyées sont forcées à 0 et, pour la seconde, elles sont forcées à 1. Cela permet de s'assurer de détecter n'importe quelle erreur qui pourrait avoir altéré la partie donnée de 64 bits.

3.3.4 Limitation de la méthode

Lors de la vérification au niveau logiciel, les paquets « test » transitent d'une ressource à une autre et c'est à ces points qu'ils sont vérifiés. Pour s'assurer que les paquets arrivent à destination, les champs de l'en-tête doivent être bien assignés et ces champs restent constants lors de toutes les vérifications. Cela fait donc de l'en-tête la partie du paquet la plus à risque où des fautes pourraient ne pas être détectées. Certains champs sont plus critiques que d'autres, comme expliqué ci-dessous.

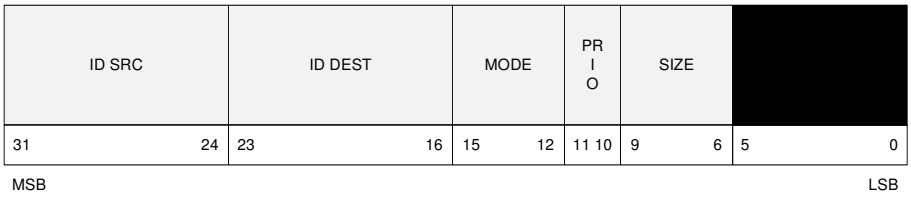


Figure 3-7: Format de l'en-tête

Source (champ ID SRC) : Une faute sur la source peut être critique lors de l'exécution de l'application, car plusieurs paquets peuvent ne pas être considérés s'il y a vérification de la source. Par exemple, la partie LSB peut être plus critique que la partie MSB si le nombre de nœuds est relativement petit. Si une faute n'est pas détectée dans la partie MSB car elle force un bit à 0, il se peut qu'elle n'ait aucun impact au cours de l'application si le nombre de nœuds est petit et peut être codé sur quelques bits LSB seulement. Pour les mêmes conditions, une erreur modifiant un bit à 1 dans la partie MSB a plus de chance d'être détectée qu'un bit modifié à 0.

Destination (champ ID DEST) : Une faute sur la destination peut être très critique lors de l'exécution de l'application, car plusieurs paquets peuvent être perdus, ce qui peut entraîner l'arrêt total de l'application. Tout comme pour la source, la partie LSB est plus critique que la

partie MSB si le nombre de nœuds est relativement petit et une erreur modifiant un bit à 1 dans la partie MSB a plus de chance d'être détectée qu'un bit modifié à 0.

Mode (champ MODE) : Outre les trois modes qui sont utilisés dans cette recherche (application, test et reconfiguration), ce champ sert à identifier l'action à poser avec le paquet reçu. Si ce champ est altéré, cela peut entraîner des complications majeures au point de vue de l'application.

Priorité (champ PRIO) : Une faute qui a lieu sur la priorité n'est pas critique, car celle-ci est prise en compte lors de l'injection dans le réseau. Donc, même s'il y a corruption lors du transit, cela ne modifie en rien l'exécution.

Taille des données (champ SIZE) : Si la grandeur du paquet est modifiée, cela peut être très critique car, lors de l'extraction, trop ou pas assez de données peuvent être extraites. Cela peut également engendrer une désynchronisation pour tous les autres paquets.

Inutilisés : Les derniers bits de l'en-tête sont inutilisés. Alors, une modification sur ceux-ci ne change en rien le fonctionnement de l'application.

Tableau 3-3: Niveau critique des champs de l'en-tête

Champ	Niveau de criticité
Source	Critique
Destination	Très critique
Mode	Critique
Priorité	Non-critique
Taille des données	Très critique
Inutilisés	Non-critique

3.4 Technique de prévention des fautes au niveau matériel

Le principe de la prévention reste le même que celui présenté précédemment, c'est-à-dire qu'il est séparé en deux principales étapes : la vérification et la reconfiguration. Contrairement à la technique de prévention des fautes au niveau logiciel, la technique au niveau matériel est intégrée totalement à l'architecture du RoC autant pour la détection des fautes que pour la reconfiguration du réseau. Autre que cette particularité principale, cette méthode permet de gérer plus de types de fautes, notamment les fautes transitoires qui ne sont pas prises en charge par la technique logicielle et, également, de vérifier tous les bits des paquets et même ceux de l'en-tête.

3.4.1 Types de fautes

Les différents types de fautes, permanentes et transitoires, ont déjà été présentés en détail dans la section *1.2.1 Causes et classification des fautes.*

Les fautes représentées ici comme étant des **fautes multiples** représentent le fait qu'il se peut fort bien que lorsque des fautes sont présentes, elles ne touchent pas qu'un seul bit. Si nous pensons seulement à un changement du champ magnétique, c'est beaucoup plus un groupe de bits qui est touché qu'un bit spécifique.

3.4.2 Modifications apportées à l'architecture du RoC

Les modifications apportées pour la technique précédente sont conservées, notamment l'ajout du module *bitmap_error*, la vérification du mode des paquets et la reconfiguration des canaux. Par contre, certaines modifications supplémentaires ont dû être effectuées au niveau du module *node* pour permettre que la prévention des fautes soit totalement prise en charge par l'architecture du RoC.

3.4.2.1 **Modifications au module *node***

Les modifications qui ont été apportées au module *node*, lors de l'élaboration de la méthode de prévention au niveau logiciel, pour permettre la reconfiguration des canaux sont également valables pour la technique au niveau matériel. Pour plus de détails sur ces modifications, se référer à la section *3.3.2.1 Modifications au module node*. Par contre, étant donné que la méthode de prévention des fautes au niveau matériel est intégrée en totalité à l'architecture du réseau comme son nom l'indique, des ajouts supplémentaires doivent être apportés. Toutes les modifications apportées au module *node* de base du RoC sont présentées en hachuré et en relief sur la Figure 3-8.

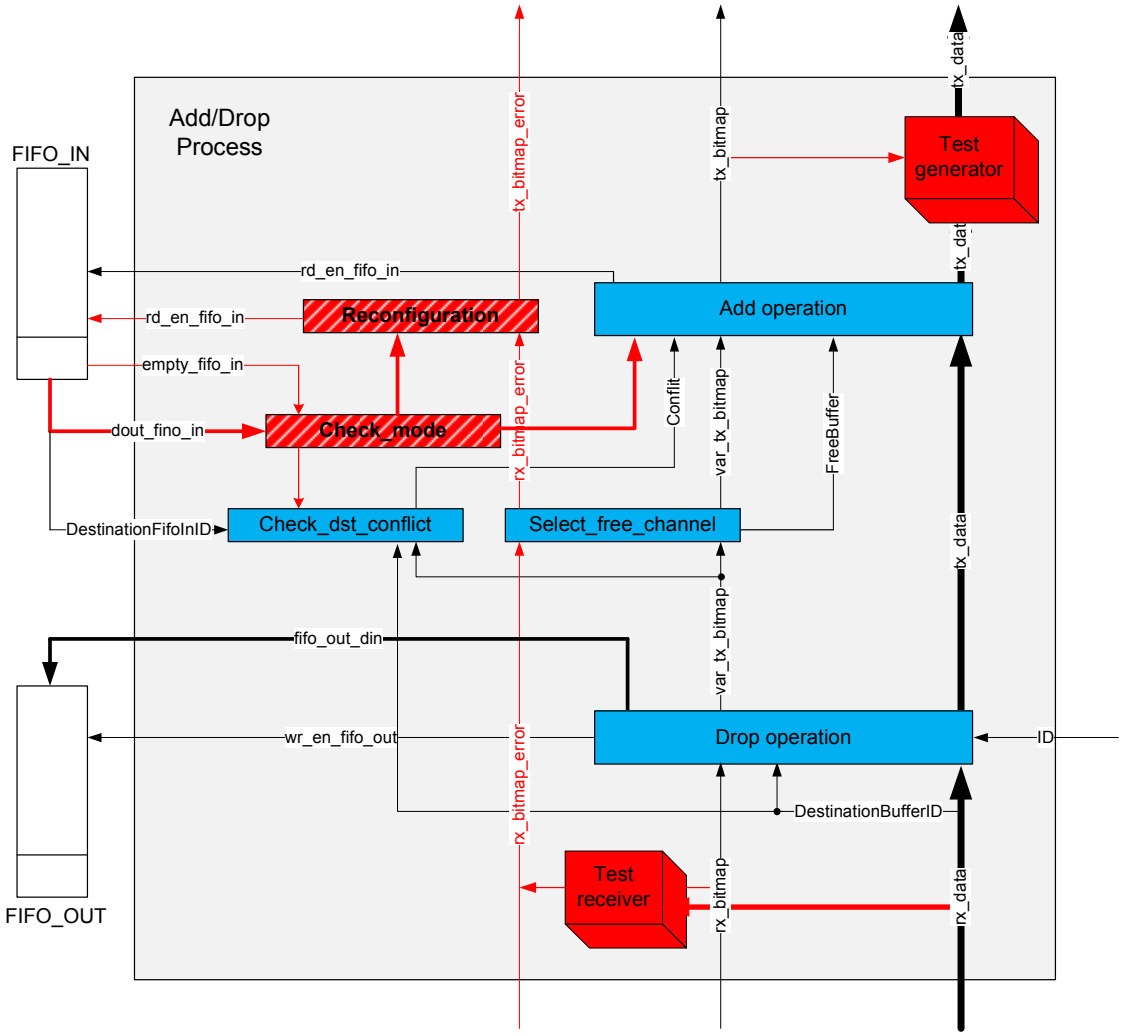


Figure 3-8: Module *node* avec intégration de la technique de prévention au niveau matériel

Le principe de la prévention se fait en trois grandes étapes au niveau matériel :

1) Vérification de l'intégrité des paquets de test et reconfiguration : Après que les paquets aient transité sur les canaux, lorsqu'ils arrivent au nœud, leur destination est vérifiée. Par contre, s'il n'y avait pas de paquets de type « application » à échanger, un paquet de type « test » a été envoyé pour permettre de vérifier l'intégrité du canal. Ces paquets sont donc transmis au module *test_receiver* pour la vérification. Si une erreur est détectée, le canal est immédiatement désactivé pour ce nœud et le sera pour la totalité des nœuds au cycle suivant.

2) Sélection du canal d'envoi des paquets applicatifs : les paquets de type applicatif déjà présents sur les canaux du RoC poursuivent leur chemin. Pour ceux nouvellement injectés à l'intérieur du réseau, le canal de communication est sélectionné selon le principe du tourniquet ou par priorité de canal.

3) Envoi des paquets de test : Après l'ajout du paquet pouvant provenir de la FIFO_IN et des paquets provenant du nœud précédent destinés à un autre nœud, des paquets de type « test » générés par le module *test_generator* sont envoyés sur les autres canaux non utilisés dans le but de vérifier leur intégrité.

3.4.2.2 **Ajout du module *test_generator***

Le module *test_generator* génère les paquets de test qui sont envoyés sur les canaux non utilisés pour vérifier leur intégrité. Le tableau ci-dessous décrit l'interface du module.

Tableau 3-4: Description de l'interface du module *test_generator*

Port	Entrée/Sortie	Description
Clk	Entrée	Horloge pour la synchronisation.
Resest	Entrée	Réinitialise le module.
test_packet	Sortie	Paquet de test généré.
packet_ready	Sortie	Indique si le paquet de test est prêt à être envoyé.

Les paquets de test sont générés de façon aléatoire à chaque cycle pour permettre une meilleure étendue de la vérification. Le module est implémenté de deux façons pour effectuer la détection des fautes : à l'aide d'un code CRC-16 ou avec un bit de parité.

Dans le cas du code CRC, une donnée de 80 bits est générée aléatoirement et le code CRC-16 associé est intégré au paquet pour compléter les 96 bits. Pour ce faire, un module *crc_16* a été introduit au *test_generator* pour permettre le calcul du code. Quand le paquet est complété, le signal *packet_ready* est mis à 1. Pour la méthode de détection avec le bit de parité, le principe est le même. Par contre, c'est une donnée de 95 bits qui est générée aléatoirement et le bit restant est comblé par le calcul de la parité.

3.4.2.3 **Ajout du module *test_receiver***

Comparativement au module *test_generator*, le module *test_receiver* est un peu plus complexe. Il permet de vérifier l'intégrité des canaux et de gérer la reconfiguration de ceux-ci. Le tableau ci-dessous décrit l'interface du module.

Tableau 3-5: Description de l'interface du module *test_receiver*

Port	Entrée/Sortie	Description
clk	Entrée	Horloge pour la synchronisation.
resest	Entrée	Réinitialise le module.
rx_data	Entrée	Vecteur contenant les paquets présents sur les canaux à l'entrée du nœud.
rx_bitmap	Entrée	Vecteur indiquant quel type de paquet transite sur le canal (1 pour applicatif, 0 pour test).
channel	Sortie	Le canal qui est vérifié.
error	Sortie	Le statut d'erreur à envoyer pour le canal (1 pour fautif, 0 pour valide).
enable	Sortie	Signal indiquant si les sorties du module sont valides.

Le module *test_receiver* fonctionne principalement en trois étapes : la sélection du canal à vérifier, la validation des données et la gestion des fautes.

Lorsque les paquets sont reçus du nœud précédent, la sélection du canal à tester se fait selon un principe de tourniquet. Si un paquet de test se trouve sur le canal choisi, il y aura vérification des données. Sinon, le canal est incrémenté pour le prochain cycle et il n'y a pas de vérification à ce cycle. Cette méthode de sélection permet d'augmenter les chances que tous les canaux soient validés au cours de l'exécution.

Par la suite, on procède à la validation des données du paquet de test pour vérifier l'intégrité du canal. Le module est implémenté soit avec la méthode de détection à l'aide d'un code CRC-16 ou celle avec un bit de parité. Dans le cas du code CRC, la vérification nécessite deux cycles. Ceci est dû au code CRC-16 qui doit être calculé avec les données reçues et être comparé avec le code reçu, ce qui prend un cycle. Pour la méthode de détection avec le bit de parité, un seul cycle est nécessaire. Lorsque le paquet est reçu, la parité des données est recalculée et comparée avec celle reçue.

Finalement, si une erreur est détectée, elle est ajoutée à un compteur de fautes pour permettre de déterminer si la faute doit être prise en charge ou pas selon la technique de gestion de fautes utilisée. Si le canal doit être désactivé ou réactivé (dans le cas où une faute ne serait pas détectée), les signaux *channel*, *error* et *enable* sont fixés aux valeurs voulues. Les techniques de gestion de fautes, qui seront décrites plus en détail à la section suivante, permettent de prendre en charge les fautes transitoires en plus des fautes permanentes.

3.4.3 Techniques de vérification

Contrairement à la technique de prévention au niveau logiciel qui se déroule seulement au début de l'exécution, la technique de prévention au niveau matériel se déroule tout au long de l'exécution et simultanément à l'application. Par contre, le but de la technique reste le même : tout d'abord, détecter les fautes présentes sur le réseau et, par la suite, reconfigurer le réseau dans le but d'éviter la propagation des erreurs. Cette méthode permet de gérer les fautes permanentes et transitoires et de détecter les fautes sur la totalité du paquet incluant l'en-tête.

De façon générale, lorsque des canaux ne transportent pas des paquets de type « applicatif », des paquets de type « test » sont envoyés sur ceux-ci. Lors de la réception des paquets, un canal est sélectionné pour être vérifié et ses données sont validées. Selon le résultat obtenu, la reconfiguration du réseau est effectuée selon la technique de gestion des fautes utilisée. Le tableau suivant démontre plus clairement la procédure.

Tableau 3-6: Algorithme général de la technique de prévention au niveau matériel

#	Étape
1	Les paquets provenant des FIFO_IN n'étant pas en conflit sont injectés dans le réseau.
2	Si des canaux ne sont pas occupés par des paquets « application », des paquets « test », générés par le *test_generator*, sont envoyés vers le nœud suivant.
3	Le nœud suivant reçoit les paquets et extrait ceux rendus à destination.
4	Simultanément à l'étape #3, les paquets « test » sont appliqués à l'entrée du module *test_receiver*.
5	Le module *test_receiver* sélectionne le paquet « test » à vérifier en fonction du principe du tourniquet.
6	Si une erreur est détectée, le module *test_receiver* commande de désactiver le canal testé.
7	Si aucune erreur n'est détectée et que le segment est déjà désactivé, il y a vérification du compteur de fautes pour savoir s'il doit être réactivé (cas d'une faute transitoire).
8	Suite à la reconfiguration des canaux, l'injection des paquets provenant des FIFO_IN reprend, de même que l'envoi des paquets « test ».

Finalement, ce travail évalue cette technique selon différents paramètres dans le but d'obtenir la meilleure efficacité. Les différentes simulations permettent de comparer :

La sélection des canaux pour l'envoi des paquets : Procéder par priorité de canal pour l'envoi d'un nouveau paquet « applicatif », comme il est implémenté dans l'architecture de base du RoC ou utiliser un principe de tourniquet pour distribuer le trafic de façon plus équivalente sur tous les canaux.

Les mécanismes de détection des fautes : L'utilisation d'un code CRC-16 versus un bit de parité.

Les algorithmes de gestion des fautes : On peut catégoriser les deux algorithmes testés de reconfiguration instantanée et de reconfiguration après *X* occurrences. Dans le cas de la reconfiguration instantanée, dès qu'une faute est détectée, le canal est désactivé et est réactivé seulement lorsqu'il aura été testé *X* fois valide par ce nœud, ce qui donnera le statut de transitoire à la faute. Pour ce qui est de la reconfiguration après *X* occurrences, le canal est désactivé quand il a été détecté fautif à *X* reprises et est réactivé lorsqu'il a été détecté valide à *Y* reprises.

CHAPITRE 4
ENVIRONNEMENT DE TEST

Pour permettre de comparer les différentes techniques de prévention des fautes et leurs paramètres dans le but de déterminer celle qui assure une meilleure performance, un environnement de test permettant l'injection des fautes a dû être créé.

4.1 Injection de fautes

Un module est dédié à l'injection des fautes dans le système. Ce module est essentiel pour tester la fiabilité d'une technique de tolérance aux fautes. Les fautes sont modélisées au niveau logique sur le chemin de données.

4.1.1 Format des fautes

Les fautes sont injectées à l'aide d'un mot de 32 bits dont le format est présenté à la Figure 4-1. Chacun des champs est décrit plus en détails dans le Tableau 4-1.

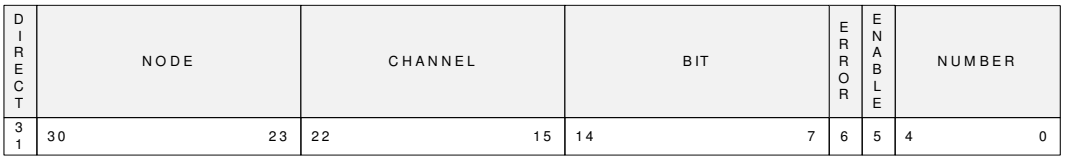


Figure 4-1: Format d'un paquet de faute

Tableau 4-1: Paramètres d'une faute

Paramètre	Description
Direction (d)	Horaire (0) ou antihoraire (1)
Nœud (n)	La faute sera effective à la sortie du nœud *n* dans la direction *d* choisie. (Entre 0 et NB_NODES-1)
Canal (c)	La faute sera effective sur le canal *c* choisi pour le nœud *n*. (Entre 0 et NB_CHANNELS-1)
Bit (b)	La faute sera effective pour le bit *b* choisi sur le canal *c*. (Entre 0 et DATA_PATH-1)
Erreur	La faute à mettre sur le bit *b* choisi. (*stuck-at* 0 ou 1)
Actif	Si la faute sur le bit *b* doit être activée (1) ou désactivée (0).
Nombre	Nombre de bits à mettre en faute à partir du bit *b* choisi.

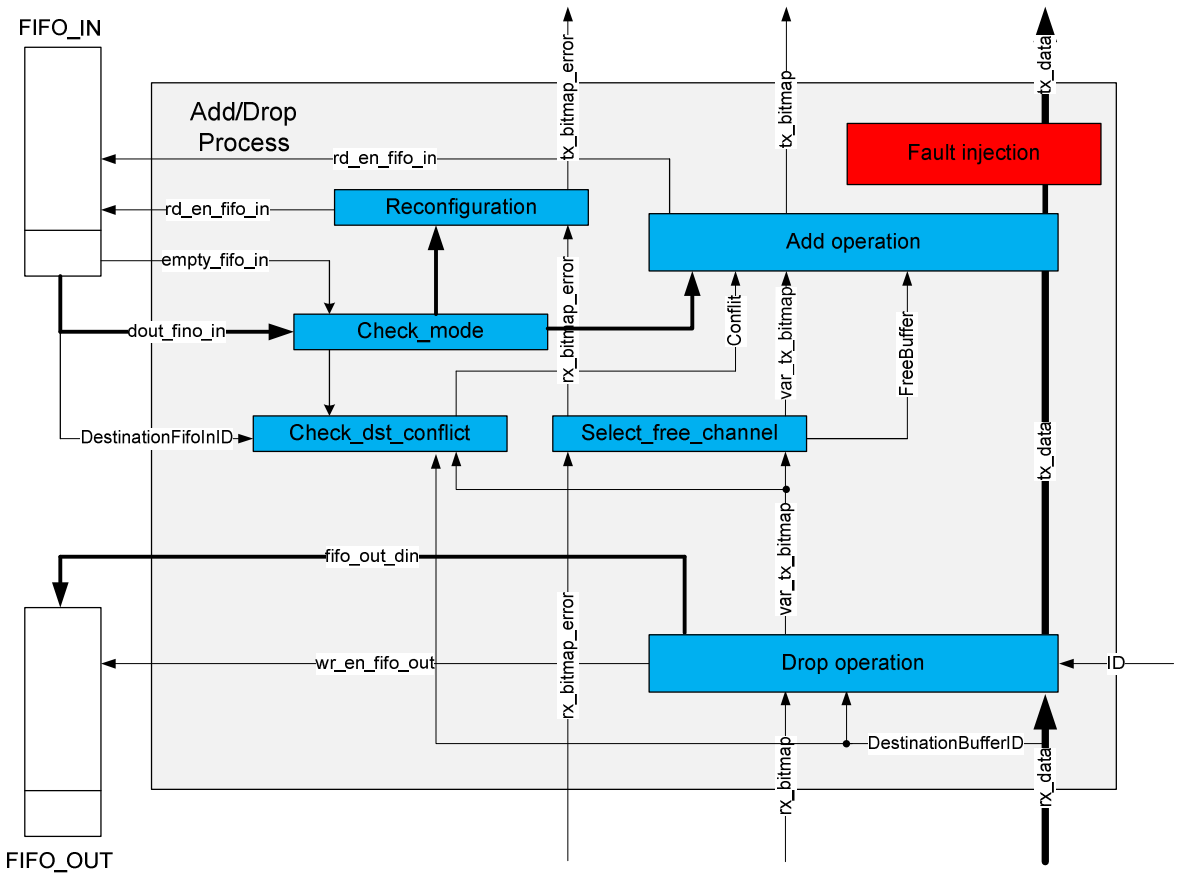


Figure 4-2: Emplacement du processus d'injection de faute dans un nœud (exemple d'un nœud supportant la technique de vérification logicielle)

Chacune des fautes est injectée sur le chemin de données à la sortie du nœud *n* choisi comme il est présenté sur la Figure 4-2. Donc, on s'attend à ce que la faute soit détectée par le nœud suivant. Les fautes injectées sont de type valeur forcée (*stuck-at*), donc le bit choisi est forcé à la valeur déterminée (0 ou 1). Le paramètre « Actif » permet de modéliser les fautes transitoires qui surviennent pendant un court laps de temps; lors de la première injection de la faute, celle-ci est activée et, après le nombre de cycles voulus, il suffit de renvoyer les mêmes paramètres pour la faute en indiquant qu'elle doit être désactivée. Également, le paramètre « Nombre » permet de mettre en faute plusieurs bits successifs simultanément dans le but de simuler des perturbations qui influenceraient une région plutôt qu'un simple bit.

4.2 Architecture des bancs de tests

L'environnement de test a été élaboré à l'aide de l'*Open Verification Methodology* (OVM) (Glasser, 2009). L'intérêt d'un tel standard est qu'il permet de scinder l'architecture d'un banc de test en plusieurs blocs fonctionnels indépendants les uns des autres. Chacun de ces blocs est dédié à une seule tâche et ils sont indépendants du système lui-même. De cette façon, il est possible de modifier les blocs existants ou d'en rajouter de nouveaux sans avoir à réadapter le système en entier à chaque fois. L'architecture générale des bancs de tests qui est utilisée ici est celle qui est présentée à la Figure 4-3 et qui correspond à une représentation allégée d'une organisation de bancs de tests concentrique. En effet, étant donné les vérifications à apporter au système, il n'est pas nécessaire d'utiliser tous les blocs de l'architecture OVM.

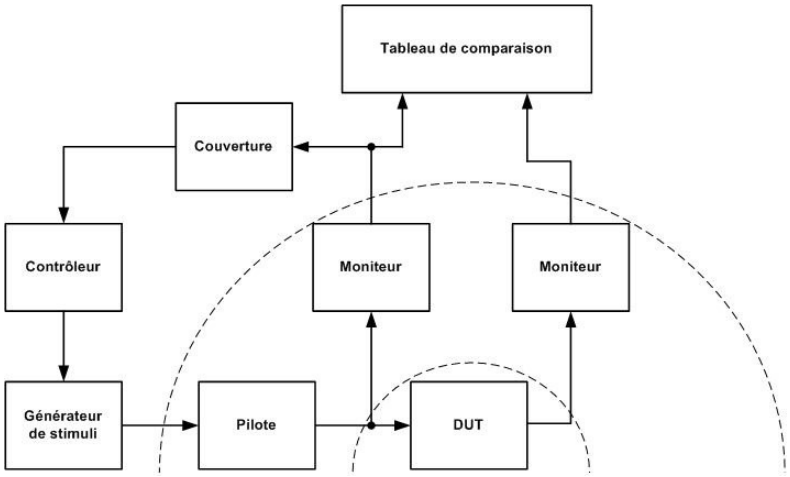


Figure 4-3: Architecture générale du banc de tests

Les modules du banc de tests ont été implémentés au niveau RTL. La Figure 4-4 représente l'architecture spécifique du banc de tests utilisé pour vérifier et comparer les différentes techniques de tolérance aux fautes.

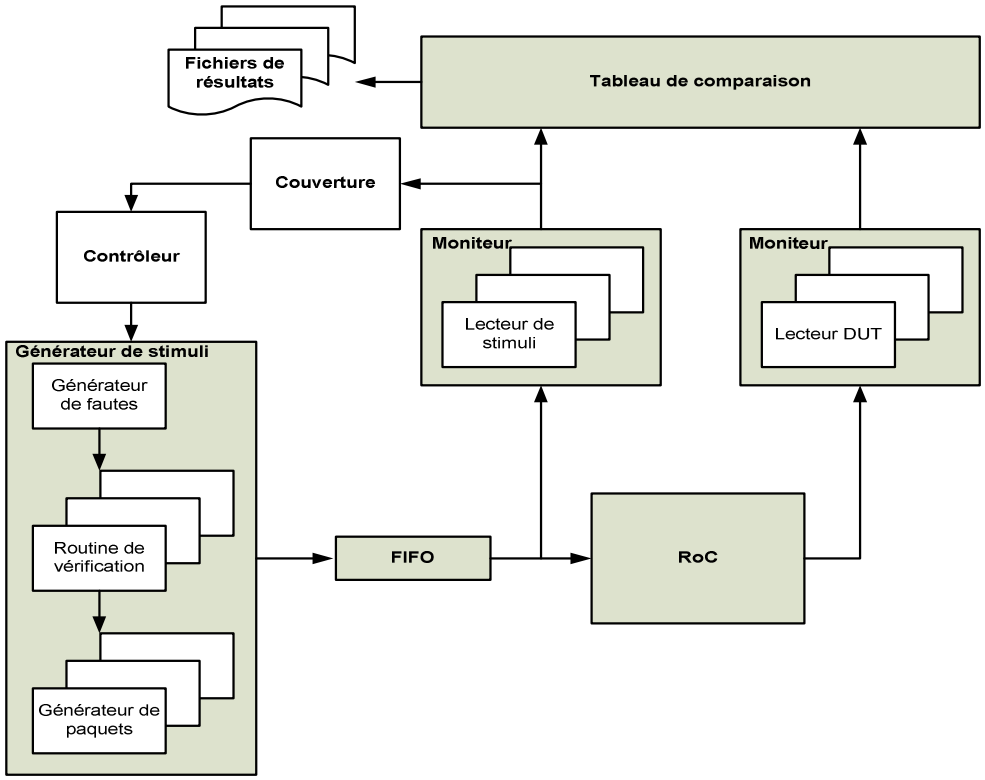


Figure 4-4: Architecture spécifique du banc de tests

Générateur de stimuli : Le générateur de stimuli comporte deux ou trois parties en fonction de la technique de tolérance aux fautes à tester. Dans le cas de la technique de vérification logicielle, les trois modules sont essentiels : le générateur de fautes, la routine de vérification et le générateur de paquets, tel qu'illustré sur la Figure 4-4. Par contre, pour la technique de vérification matérielle, le module pour la routine de vérification n'est pas nécessaire.

> ***Générateur de fautes*** **:** Le générateur de fautes permet l'injection de fautes dans le réseau dans le but de tester les techniques de tolérance aux fautes. Le format des fautes ainsi que le fonctionnement plus spécifique de l'injection des fautes sont détaillés à la section suivante. Dans le cas de fautes permanentes, le générateur de fautes injecte les fautes et, par la suite, envoie un signal de départ au module de routine de vérification. Par contre, dans le cas de fautes transitoires, le générateur de fautes procède à l'injection des fautes tout au long de l'exécution de la simulation.

***Routine de vérification* :** La routine de vérification comprend *N* sous-modules où *N* représente le nombre de nœuds du RoC. Un nœud est considéré maître, il commande les reconfigurations et initie les vérifications, et les autres sont considérés esclaves. L'implémentation de la routine correspond aux différentes variantes de la technique de vérification logicielle énoncées à la section *3.3.3 Technique de vérification*. Lorsque la routine de vérification est terminée, un signal est transmis au générateur de paquets pour qu'il débute l'envoi.

***Générateur de paquets* :** Le générateur de paquets simule en quelque sorte l'application qui serait sur les différents nœuds du RoC. Les différents scénarios sont présentés à la section *4.3 Description des scénarios de simulation*.

RoC : Le RoC est le design sous test (*DUT*). Les stimuli lui sont donc appliqués et ses résultats sont analysés.

Moniteur : Les moniteurs permettent de récupérer les données comparatives. Dans ce cas-ci, ce sont les paquets entrant dans le RoC et les paquets en sortant.

Lecteur de stimuli : Le lecteur de stimuli permet de récupérer les paquets entrant dans le RoC.

Lecteur DUT : Le lecteur DUT permet de récupérer les paquets sortant du RoC.

Tableau de comparaison : Le tableau de comparaison, comme son nom l'indique, permet de comparer les paquets reçus avec les paquets envoyés. Le tableau de comparaison génère trois fichiers de résultats permettant d'analyser les différentes techniques de tolérance aux fautes.

1) Enregistrer tous les paramètres des fautes injectées en plus de leur cycle d'envoi.
2) Enregistrer tous les paramètres des paquets qui ont été envoyés et bien reçus en plus de leurs cycles d'envoi et de réception. Grâce à ces données, on obtient le calcul du délai moyen des paquets sur le RoC.
3) Enregistrer les paquets reçus qui diffèrent de ceux envoyés.

4.3 Description des scénarios de simulation

4.3.1 Scénarios d'injection des fautes

Tel que mentionné précédemment, deux types de faute sont principalement modélisés : les fautes permanentes et les fautes transitoires.

Fautes permanentes

Les fautes permanentes sont décrites plus en détail à la section *1.2.1 Causes et classification des fautes*. Trois différents scénarios reliés à ces fautes sont étudiés.

1- Injection de fautes permanentes simples : Un seul bit d'un seul segment par canal est en faute et la faute est activée tout au long de l'exécution.

2- Injection de fautes permanentes simples sur plusieurs bits : Plusieurs bits d'un seul segment par canal sont en faute et les fautes sont activées tout au long de l'exécution.

3- Injection de fautes permanentes multiples : Plusieurs bits sur plusieurs segments sont en faute tout au long de l'exécution. Contrairement aux autres scénarios, il peut y avoir plusieurs segments par canal en faute.

Fautes transitoires

Les fautes transitoires sont décrites plus en détail à la section *1.2.1 Causes et classification des fautes*. Trois scénarios similaires à ceux traitant des fautes permanentes sont analysés :

1- Injection de fautes transitoires simples : Un seul bit est en faute à la fois sur tout le réseau. Chaque faute dure entre 1 et *X* cycles et il y a un délai de 1 à *Y* cycles entre la fin de la faute et l'injection de la suivante.

2- Injection de fautes transitoires simples sur plusieurs bits : Plusieurs bits d'un même segment sont en faute simultanément. Chaque faute dure entre 1 et *X* cycles et il y a un délai de 1 à *Y* cycles entre la fin de la faute et l'injection de la suivante.

3- Injection de fautes transitoires multiples : Plusieurs bits sur plusieurs segments peuvent être en faute simultanément. Chaque faute dure entre 1 et *X* cycles.

4.3.2 Scénario de communication

Communications aléatoires

Pour un réseau ayant une topologie en anneau bidirectionnelle, une des distributions favorisant le rendement du réseau est celle où les communications sont distribuées aux ressources avoisinantes selon un pourcentage favorisant la proximité. Par contre, dans le but de tester le réseau dans des conditions un peu plus contraignantes, la distribution choisie consiste à envoyer des messages à des destinations aléatoires. De cette façon, chaque nœud communique avec tous les autres au hasard.

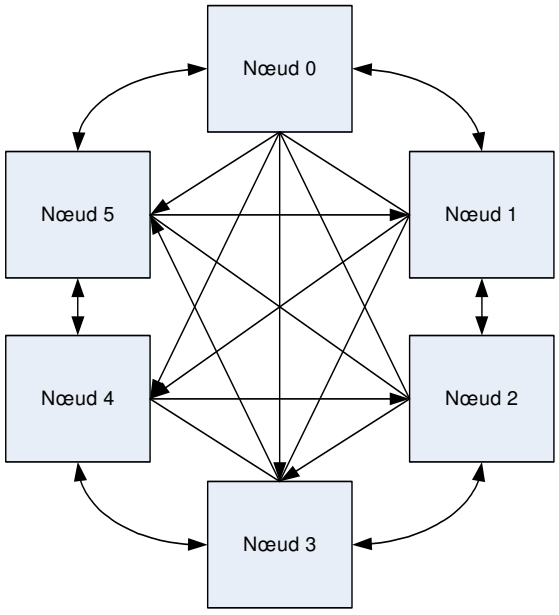
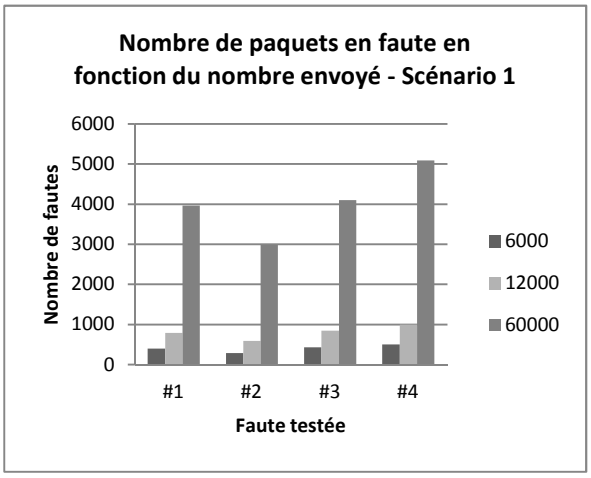


Figure 4-5: Scénario de communications aléatoires

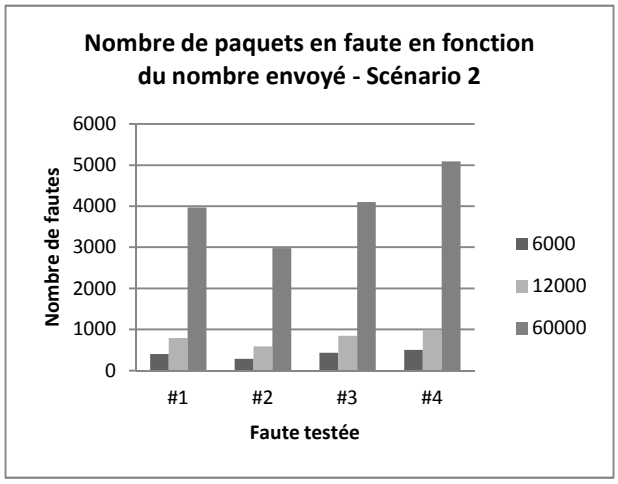
4.4 Hypothèses

Certaines hypothèses ont été posées et vérifiées dans le but d'alléger les simulations.

1) Le nombre de paquets reçus en faute augmente proportionnellement avec le nombre de paquets envoyés. De cette façon, il est possible d'envoyer un plus petit nombre de paquets dans le but de diminuer le temps de simulation et de transposer les résultats à différentes échelles. Les graphiques ci-dessous (**Erreur ! Source du renvoi introuvable.**) confirment cette hypothèse en présentant deux cas d'envois aléatoires de paquets testés avec quatre fautes différentes. Par exemple, pour la première faute dans le premier scénario, nous obtenons 403 paquets erronés pour un envoi de 6 000 paquets, 794 paquets erronés pour 12 000 envoyés et 3 995 paquets erronés pour 60 000 envoyés. En faisant le calcul, nous nous apercevons que le rapport est le même pour le nombre de paquets envoyés et le nombre de paquets erronés reçus. La même corrélation est présentée avec le second scénario qui présente les résultats avec des fautes affectant des bits différents. À la suite de ces résultats, nous avons décidé d'utiliser des envois de 10 000 paquets par nœud permettant d'avoir un temps de simulation raisonnable et d'obtenir des ensembles de résultats de simulation assez complets.



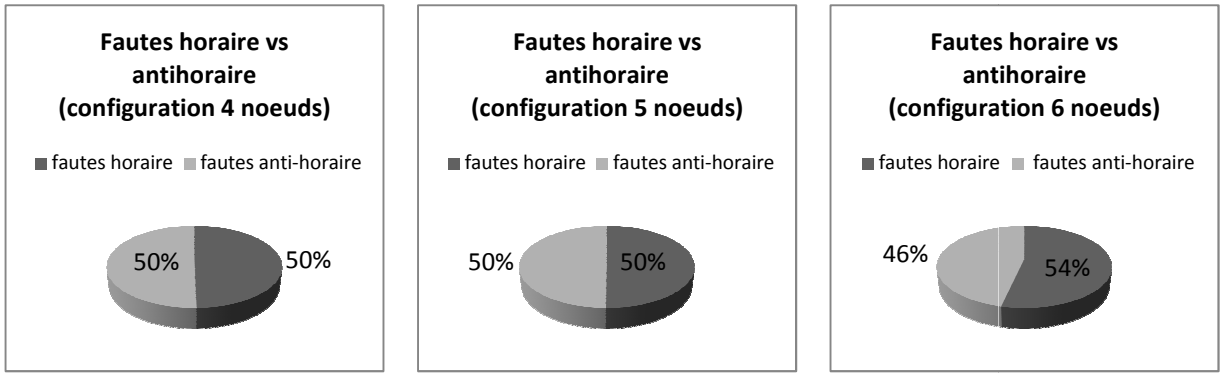
a) Scénario aléatoire #1



b) Scénario aléatoire #2

Figure 4-6: Graphiques démontrant la proportionnalité des résultats

2) Dans une distribution aléatoire, une faute dans le sens horaire n'est pas plus dommageable qu'une faute dans le sens antihoraire. L'utilisation de la génération aléatoire des fautes est alors indiquée. Peu importe le sens de la faute injectée, elle aura le même poids sur le fonctionnement. Les graphiques ci-dessous (**Erreur ! Source du renvoi introuvable.**) montrent le pourcentage des paquets erronés reçus en fonction de la direction de la faute.



a) Configuration 4 nœuds **b) Configuration 5 nœuds** **c) Configuration 6 nœuds**

Figure 4-7: Graphiques présentant le taux de répercussion des fautes horaire et antihoraire

CHAPITRE 5
RÉSULTATS ET ANALYSE

Voici les résultats qui ont été obtenus dans le but de comparer les différentes techniques de prévention. De façon générale, il y a deux grandes catégories : la prévention effectuée au niveau logiciel et celle au niveau matériel. Chacune de ces catégories a été élaborée avec différentes méthodes de détection de fautes, de sélection des canaux et, dans le cas du niveau matériel, de reconfiguration du réseau. Les différentes alternatives ont été comparées en fonction du nombre de fautes évitées, du délai moyen des paquets, du temps de vérification et du nombre de ressources utilisées.

La caractéristique de base qu'il faut pour que la méthode développée soit fonctionnelle est que le réseau auquel elle est appliquée soit multidimensionnel. En d'autres termes, il doit posséder plusieurs canaux de communication parallèles. C'est donc avec des configurations du RoC bidirectionnel disposant de plusieurs canaux que les simulations ont été effectuées à une fréquence de 100 MHz avec *Modelsim*.

5.1 Méthode de vérification logicielle

Les différentes simulations sont effectuées dans le but de définir la méthode de vérification logicielle la plus efficace. Les différents critères analysés sont notamment la sélection des canaux de communication, la technique de détection des fautes, le temps de vérification et le nombre de ressources nécessaires. Le Tableau 5-1 présente les différentes catégories de simulations effectuées en fonction de la configuration du réseau et de la technique de vérification. Toutes les simulations décrites ont été effectuées avec des envois de 10 000 paquets par nœud.

Tableau 5-1: Simulations effectuées pour la méthode de prévention des fautes au niveau logiciel

Technique / Configuration	Simple parité		Simple CRC		Double CRC	Double 0 et 1
	Prio 0	Tour.	Prio 0	Tour.		
8 nœuds et 3 canaux	X	X	X	X	X	
16 nœuds et 3 canaux			X		X	X
32 nœuds et 4 canaux						X

5.1.1 Sélection des canaux de communication

L'implémentation de base du RoC effectue la sélection des canaux de données par priorité de canal, soit en attribuant le paquet sur le premier canal libre sans conflit de destination. De cette façon, le trafic sur le canal 0 est substantiellement supérieur au trafic présent sur le dernier canal. Dans l'optique de vouloir réduire les effets de la présence d'une faute sur le réseau, la possibilité de distribuer le trafic sur tous les canaux le plus équitablement possible (à l'aide du principe du tourniquet) a également été évaluée. Les résultats obtenus pour une détection simple des fautes à l'aide du contrôle de parité sont présentés sur le graphique ci-dessous. Le nombre de paquets erronés est calculé après l'injection de cinq fautes permanentes simples (un seul bit en faute par canal, donc cinq canaux présentant des fautes) sur le réseau. Nous pouvons noter une certaine constance au niveau de la technique du tourniquet mais, comme en témoigne l'essai #2 présenté sur la Figure 5-1, cela peut aussi s'avérer défavorable dans certains cas. Par contre, ce principe permet d'assurer un minimum de paquets sans faute.

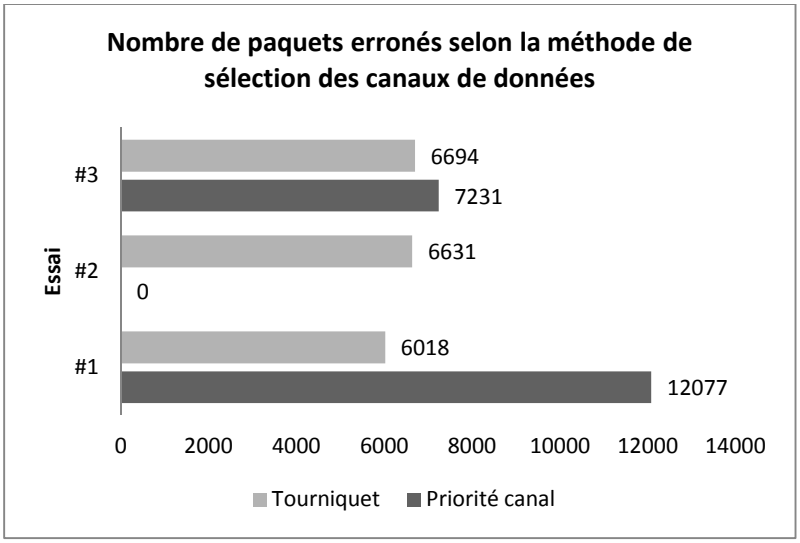
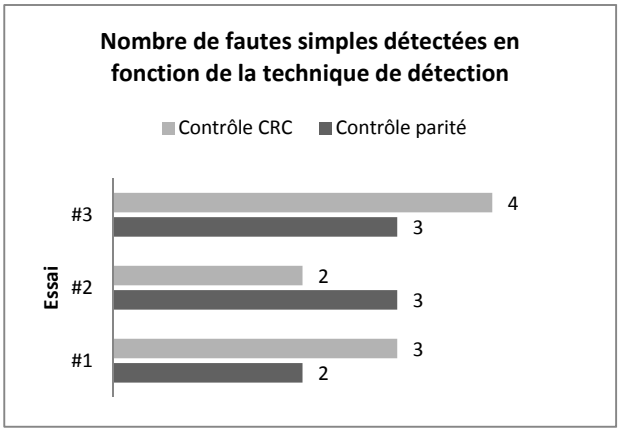


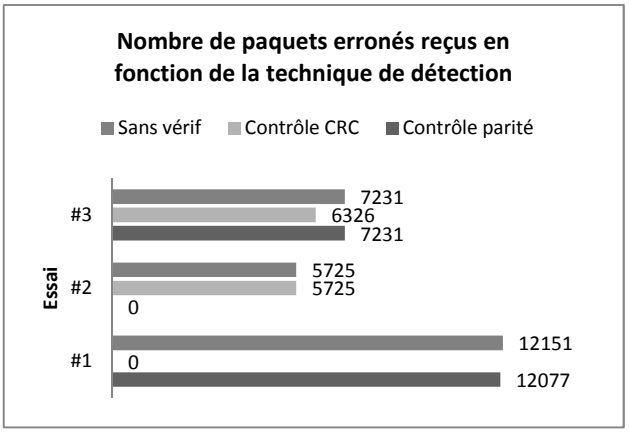
Figure 5-1: Graphique de comparaison pour la sélection des canaux

5.1.2 Technique de détection des fautes

Les deux techniques de base de détection de faute qui ont été comparées sont le contrôle de parité et le contrôle de redondance cyclique. Leur efficacité a été comparée à deux niveaux : lors d'une faute simple et lors d'une faute influençant plusieurs bits simultanément. Les graphiques ci-dessous (Figure 5-2 et Figure 5-3) présentent les résultats obtenus pour trois essais, dans le cas de fautes simples, et quatre essais, dans le cas de fautes multiples, ayant différentes fautes générées. Il y a toujours un total de cinq fautes injectées pour permettre à au moins un canal de poursuivre le traitement de l'application.

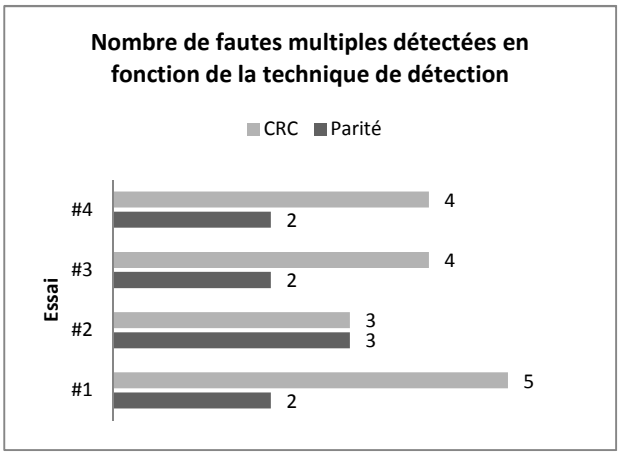


a) Nombre de fautes détectées

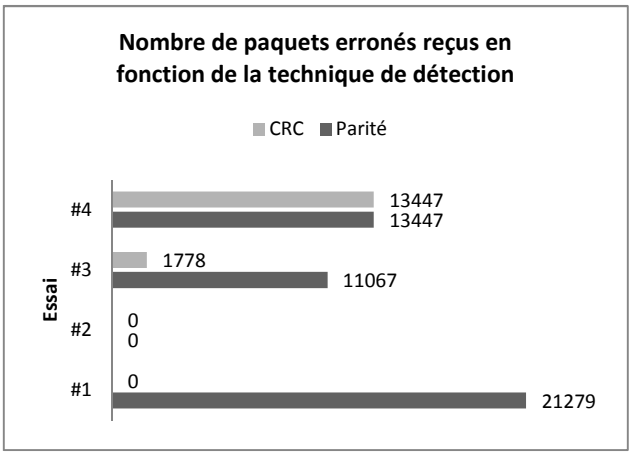


b) Nombre de paquets erronés reçus

Figure 5-2: Graphiques comparant les techniques de parité et CRC pour des fautes simples



a) Nombre de fautes détectées

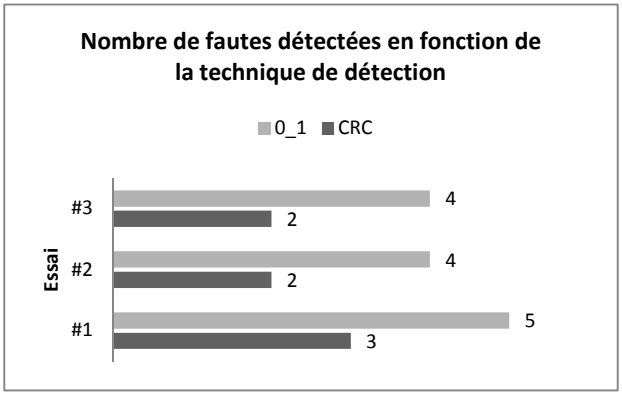


b) Nombre de paquets erronés reçus

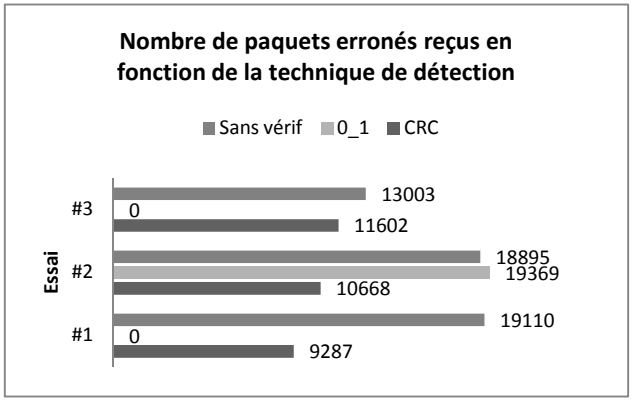
Figure 5-3: Graphiques comparant les techniques de parité et CRC pour des fautes multiples

Les résultats obtenus dans le cas de fautes simples ne sont pas très concluants. Par contre, ceux obtenus dans le cas de fautes multiples démontrent que la technique de contrôle de redondance cyclique offre de bien meilleures performances que la technique du contrôle de parité. Bien que ce ne soit pas toutes les fautes injectées qui soient détectées par une ou l'autre des techniques, nous pouvons apercevoir que certaines fois, aucun paquet erroné n'est reçu. Ceci est représenté par un 0 sur les graphiques du nombre de paquets erronés reçus en fonction de la technique de détection. Cela est principalement dû au fait que ces fautes agissent sur des champs inutilisés ou inchangés de l'en-tête (se référer à la section *3.3.4 Limitation de la méthode*).

Bien que les résultats démontrent que le contrôle de redondance cyclique offre une meilleure performance, on peut également observer que plusieurs fautes restent encore non-détectées. Ceci arrive surtout dans le cas de fautes simples, en vérifiant seulement une fois chaque segment de chaque canal. La possibilité de vérifier deux fois chaque segment avec des données de test différentes a donc été étudiée dans le but d'augmenter la performance de détection des fautes. La vérification double a également été testée sous deux variantes : un code CRC-16 et des séquences de 0 et de 1. Dans le cas du code CRC-16, deux paquets ayant des données différentes et, par le fait même, un code CRC différent, sont envoyés pour vérifier le segment. Dans le second cas, les données sont une séquence de 0 pour le premier envoi et de 1 pour le second (nommé 0_1 sur la Figure 5-4). De cette façon, une meilleure détection des fautes est assurée pour la partie des données. Les simulations ont été exécutées pour une configuration du RoC avec 16 nœuds et 3 canaux pour augmenter la charge des communications. Il y a à nouveau cinq fautes qui sont injectées. Les résultats obtenus dans le cas de fautes simples sont présentés ci-dessous.



a) Nombre de fautes détectées



b) Nombre de paquets erronés reçus

Figure 5-4: Graphiques comparant les performances des techniques de double vérification

La vérification 0_1 a démontré son efficacité. Il est à noter que toutes les erreurs non détectées dans le cas de cette vérification sont présentes sur la plage de l'en-tête. Nous pouvons remarquer, lors du second essai, que la technique de vérification 0_1 obtient un très haut nombre de paquets en faute soit 19 369. Cela est dû au fait qu'une erreur affectant un champ de l'en-tête présente sur le canal 0 n'a pas été détectée et que toutes les autres fautes l'ont été. Ceci a eu pour effet de désactiver les autres canaux et de transférer le trafic sur le canal 0 qui s'est alors vu corrompre encore plus de paquets. Le graphique ci-dessous (Figure 5-5) présente des résultats avec des fautes présentes seulement sur la partie des données et démontre une efficacité de 100% pour cette technique de détection. La configuration utilisée a 32 nœuds et 4 canaux et 7 fautes lui sont injectées.

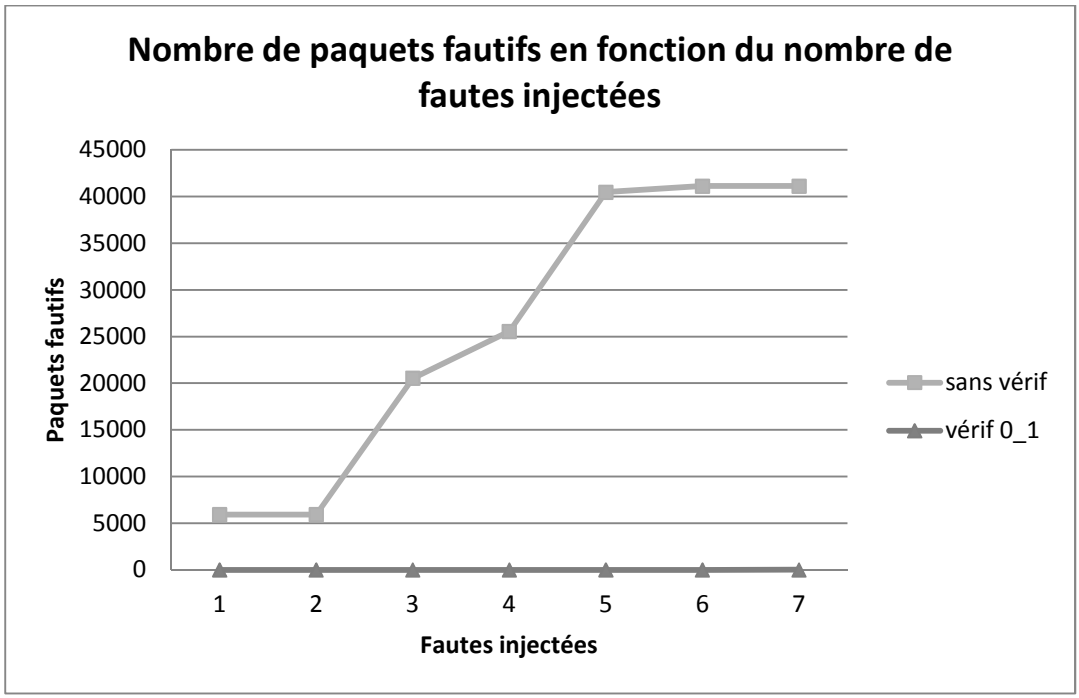


Figure 5-5: Graphique du nombre de paquets erronés avec des fautes sur la partie des données seulement

5.1.3 Délai moyen des paquets

Pour un trafic aléatoire où tous les canaux de communication sont utilisés même lorsque la sélection de canaux se fait par priorité de canal, le délai moyen des paquets augmente drastiquement lorsqu'une direction complète est désactivée. Par contre, avant qu'une direction soit complètement désactivée, le délai augmente de façon minimale lorsqu'une faute est détectée. Le délai augmente également de façon considérable lorsqu'il reste seulement un canal pour acheminer tous les paquets. La Figure 5-6 présente ce phénomène. Les fautes qui ont été injectées sont décrites dans le tableau ci-dessous et nous pouvons observer que la direction horaire est complètement désactivée après l'injection de la quatrième faute.

Tableau 5-2: Définition des fautes injectées pour tester le délai

	Fautes				
	#1	**#2**	**#3**	**#4**	**#5**
Direction	Horaire	Horaire	Antihoraire	Horaire	Antihoraire
Canal	2	1	2	0	1

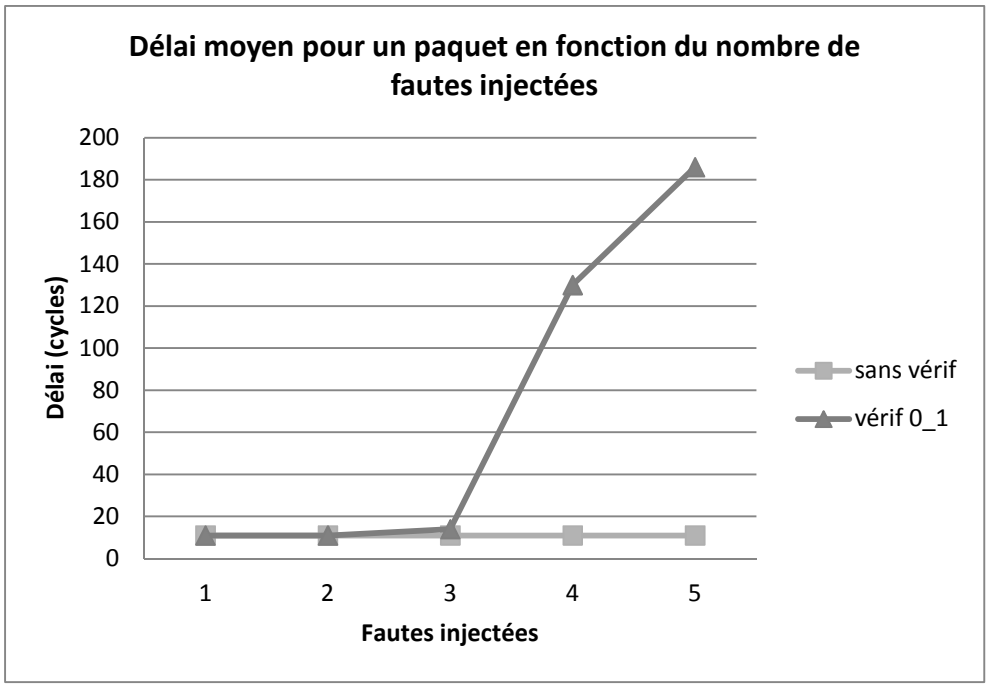


Figure 5-6: Graphique du délai moyen en fonction du nombre de fautes injectées

5.1.4 Temps nécessaire à la vérification

La vérification double a démontré son efficacité par rapport à la vérification simple grâce aux résultats obtenus précédemment. Pour atteindre ce niveau d'efficacité supérieur, il est nécessaire d'augmenter la durée de la vérification. Le temps nécessaire à la sélection des canaux à tester et à la reconfiguration reste le même, mais il double pour le temps alloué à la vérification en tant que telle. Donc, plus le nombre de nœuds augmente, plus le temps pour la sélection des canaux et la reconfiguration devient négligeable, ce qui entraîne qu'une vérification double nécessite deux fois plus de temps qu'une vérification simple. Par exemple, pour 1 024 nœuds, une vérification simple nécessite 287 200 cycles et une vérification double 573 920 cycles.

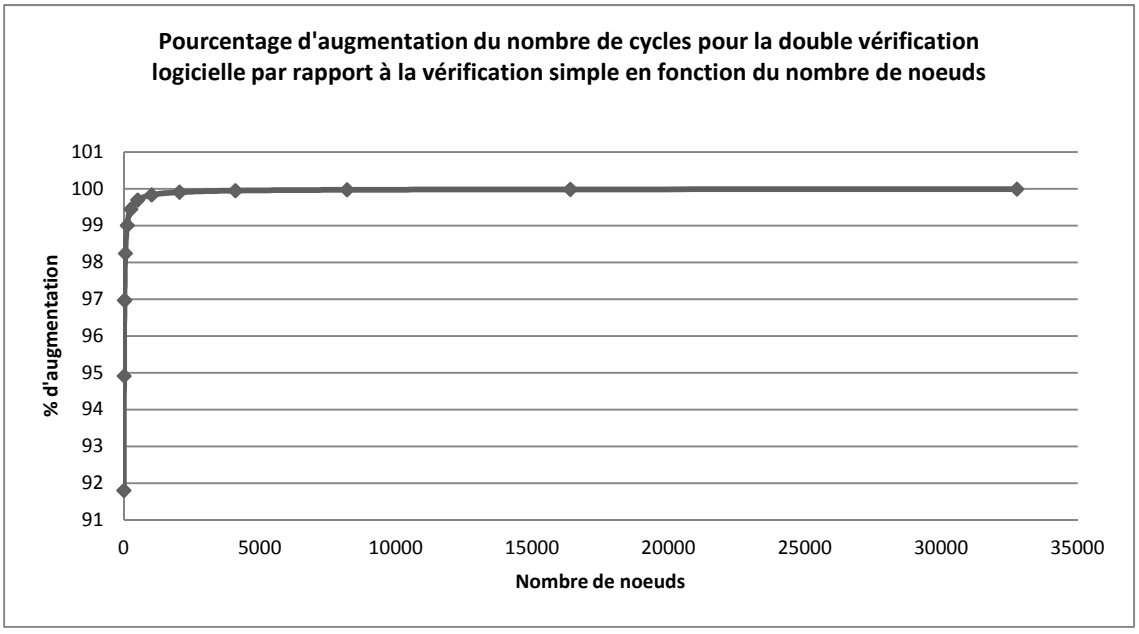


Figure 5-7: Graphique présentant l'augmentation du temps pour la vérification double

5.1.5 Prise en charge des fautes transitoires

La technique de vérification logicielle permet de détecter seulement les fautes permanentes, car elle n'a lieu qu'à l'initialisation de l'application. Même si l'on voulait appliquer cette technique périodiquement lors de l'exécution de l'application, les résultats concernant la prévention des fautes transitoires seraient loin d'être concluants compte tenu de la synchronisation qu'il faut obtenir entre l'occurrence de la faute et la vérification de celle-ci. Effectivement, la détection ayant lieu à relais prend un certain temps à se réaliser, soit $2[2C + C(2C + 28N)] + 4C$ cycles (équation 5, section *3.3.3.2*) ce qui donne 3680 cycles pour un réseau à 16 nœuds et 4 canaux par exemple. En théorie, pour la technique au niveau logiciel, lorsqu'un paquet de test se rend de la source à la destination, il transite 7 cycles à l'intérieur du réseau dans le cas du RoC. La technique de prévention prenant un temps considérable, il n'est pas souhaitable pour le bon déroulement de l'application (surtout si nous tentons de supporter une application en temps réel) de faire des récurrences de détection fréquentes. Même si l'on décidait de peut-être ralentir un peu le traitement de l'application pour permettre une meilleure tolérance aux fautes, les probabilités que les fautes transitoires soient détectées sont très faibles. Les fautes transitoires ne durant que quelques cycles, il y a très peu de chances qu'elles aient lieu en même temps que la détection des fautes sur le canal visé. Considérant une faute qui dure 100 cycles, ce qui est une durée élevée pour une faute transitoire, celle-ci a seulement une probabilité de 2,7% d'être détectée sur un réseau de 16 nœuds et 4 canaux et ce, même si elle est présente pendant la vérification (ce qui serait une chance en elle-même). La technique de prévention au niveau logiciel n'est donc pas efficace dans le cas de fautes transitoires.

5.2 Méthode de vérification matérielle

Les différentes simulations sont effectuées dans le but de définir la méthode de vérification matérielle la plus efficace autant au niveau des fautes permanentes que des fautes transitoires. Tout comme les techniques de vérification précédentes, les différents critères analysés sont notamment la sélection des canaux de communication, la technique de détection des fautes utilisée, le temps de vérification et le nombre des ressources nécessaires. Par contre, dans le cas de la vérification matérielle, nous ajoutons également les algorithmes de gestion des fautes.

5.2.1 Fautes permanentes

Comme il a été mentionné plusieurs fois depuis le début de ce mémoire, les fautes permanentes ont une présence continue tout au long de l'exécution. Pour l'implémentation de la technique de prévention au niveau matériel, la base a été les résultats obtenus au niveau logiciel. Donc, la méthode utilise le CRC comme méthode de détection des fautes, car c'est cette méthode qui a obtenu un meilleur rendement pour la vérification simple. Aussi, la sélection des canaux se fait par priorité de canal, comme il a été implémenté à la base dans l'architecture du RoC. Le Tableau 5-3 présente les différentes catégories de simulations effectuées en fonction de la configuration du réseau et de l'algorithme de la gestion des fautes. Toutes les simulations décrites ont été effectuées avec des envois de 10 000 paquets par nœud et des injections de fautes simples, sur seulement un bit.

Un fait important à soulever ici est que le réel avantage de la technique de prévention au niveau matériel est la prise en charge des fautes transitoires. C'est pourquoi les résultats de simulations concernant les fautes permanentes sont un peu moins élaborés et servent principalement de points de comparaison avec la technique logicielle.

Tableau 5-3: Simulations effectuées au niveau matériel (fautes permanentes)

Algorithme / Configuration	Reconfiguration instantanée	Reconfiguration *X* occurrences
16 nœuds et 3 canaux	X	X
32 nœuds et 2 canaux	X	

5.2.1.1 **Algorithmes de gestion des fautes**

L'algorithme de gestion des fautes exprime la façon par laquelle la reconfiguration du réseau est effectuée après qu'une faute ait été détectée. Dans le premier cas, nous avons la **reconfiguration instantanée** : dès qu'un canal est détecté comme étant en faute, il est instantanément désactivé. Il obtient donc le statut de faute permanente jusqu'à preuve du contraire i.e. lorsqu'il est vérifié et que l'intégrité de ses paquets n'est pas corrompue *X* fois consécutives. S'il est vérifié comme étant à nouveau valide, il est réactivé, ce qui donne le statut de faute transitoire. Pour le second algorithme, la **reconfiguration après *X* occurrences**, lorsqu'un canal est détecté en faute, il n'est pas automatiquement désactivé au cas où la faute ne dure qu'un court instant (cette méthode de reconfiguration serait utilisée autant dans le cas de fautes permanentes que transitoires). Il est désactivé seulement après la détection de *X* fautes consécutives sur le canal. Pour la réhabilitation du canal, le principe est le même que l'algorithme de la reconfiguration instantanée. Le canal doit être détecté *X* fois consécutives sans faute, car les fautes ne sont pas détectées à tout coup donc, de cette façon, on est assuré qu'un canal qui ne doit pas être réactivé ne le soit pas. Ici, après avoir effectué quelques essais pour trouver le nombre de vérifications idéal avant la reconfiguration, le nombre 15 s'avérait être le plus efficace en termes d'erreurs évitées et de délai. C'est ce nombre qui sera utilisé pour obtenir les résultats aux Figures 5-8 et 5-9.

Le graphique 5-8 présente les résultats obtenus pour trois différents essais effectués avec les deux algorithmes ayant chacun cinq fautes injectées de façon aléatoire. À titre comparatif, sans aucune vérification et reconfiguration, l'injection des fautes pour l'essai #1 génère 19 110 paquets en faute, l'essai #2 en génère 18 895 et l'essai #3 génère 13 003 paquets en faute. Le graphique 5-9 démontre l'évolution des paquets erronés par rapport au nombre de fautes injectées pour un essai en particulier, soit l'essai #1.

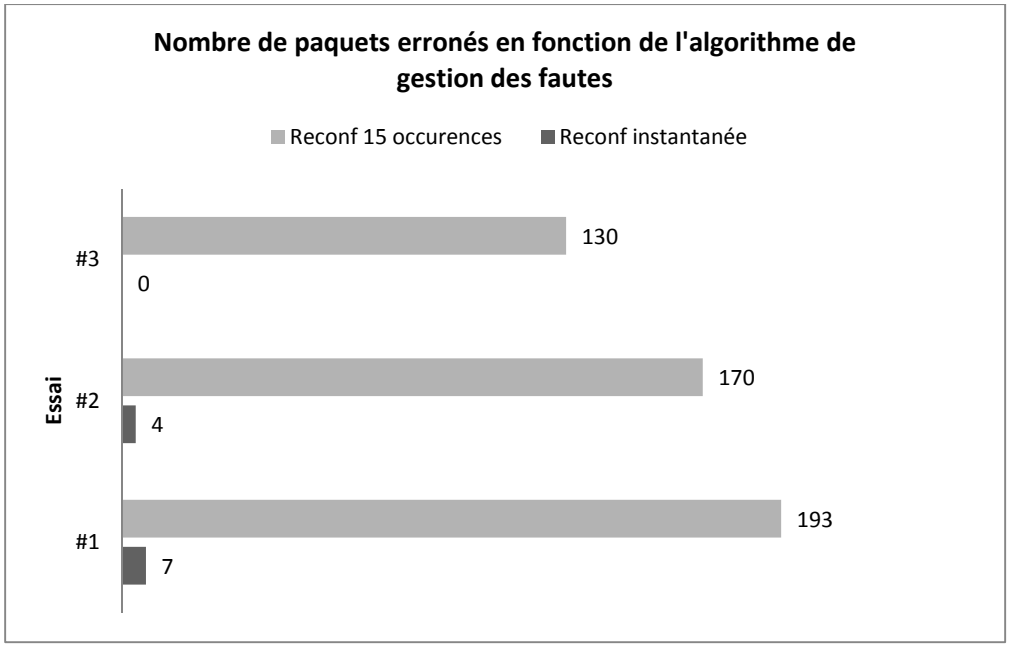


Figure 5-8: Graphique comparant les algorithmes de gestion des fautes

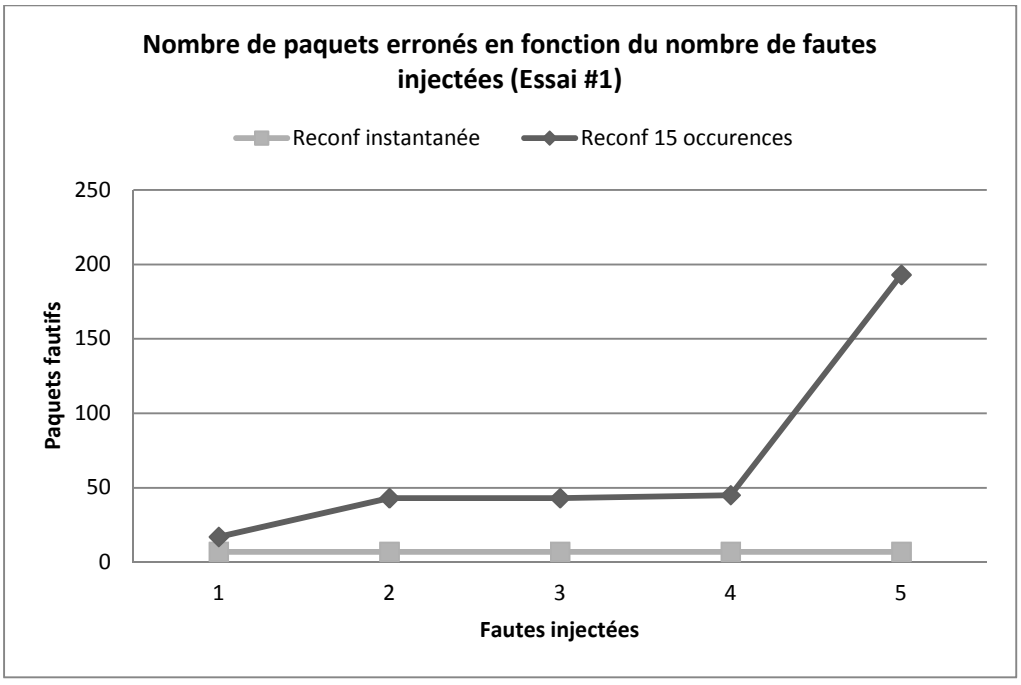


Figure 5-9: Graphique montrant l'évolution des paquets en faute

Les résultats obtenus démontrent clairement que l'algorithme de gestion des fautes utilisant la reconfiguration instantanée obtient une bien meilleure performance que celui avec une reconfiguration après 15 occurrences. Cela est dû au fait que la reconfiguration après *X* occurrences considère seulement les fautes qui durent plus de *X* cycles. Alors, les paquets transitant par le canal en faute pendant les *X* cycles avant la reconfiguration sont affectés par la faute, ce qui n'est pas le cas avec la reconfiguration instantanée. C'est pourquoi les prochains résultats de la technique de prévention des fautes au niveau matériel seront obtenus à l'aide de l'algorithme avec reconfiguration instantanée.

5.2.1.2 **Comparaison avec la technique de prévention des fautes au niveau logiciel**

Étant donné que la technique de prévention des fautes au niveau logiciel permettait seulement la détection des fautes permanentes, nous l'avons comparé avec la configuration matérielle que nous avons utilisée pour la gestion des fautes permanentes. Ici, la vérification logicielle est représentée par la vérification double 0 et 1, qui a été définie comme étant la plus efficace, et la sélection des canaux est effectuée par priorité de canal. Pour ce qui est de la vérification matérielle, elle utilise la technique de détection CRC, la reconfiguration instantanée et la sélection des canaux par priorité. Les graphiques ci-dessous (Figure 5-10) présentent les résultats obtenus en termes de fautes détectées, de nombre de paquets erronés reçus et de délai moyen des paquets pour deux essais différents ayant chacun une configuration de 16 nœuds et 3 canaux et ayant 5 fautes injectées de façon aléatoire sur 5 différents canaux. Les fautes et la distribution aléatoire sont différentes pour les deux essais.

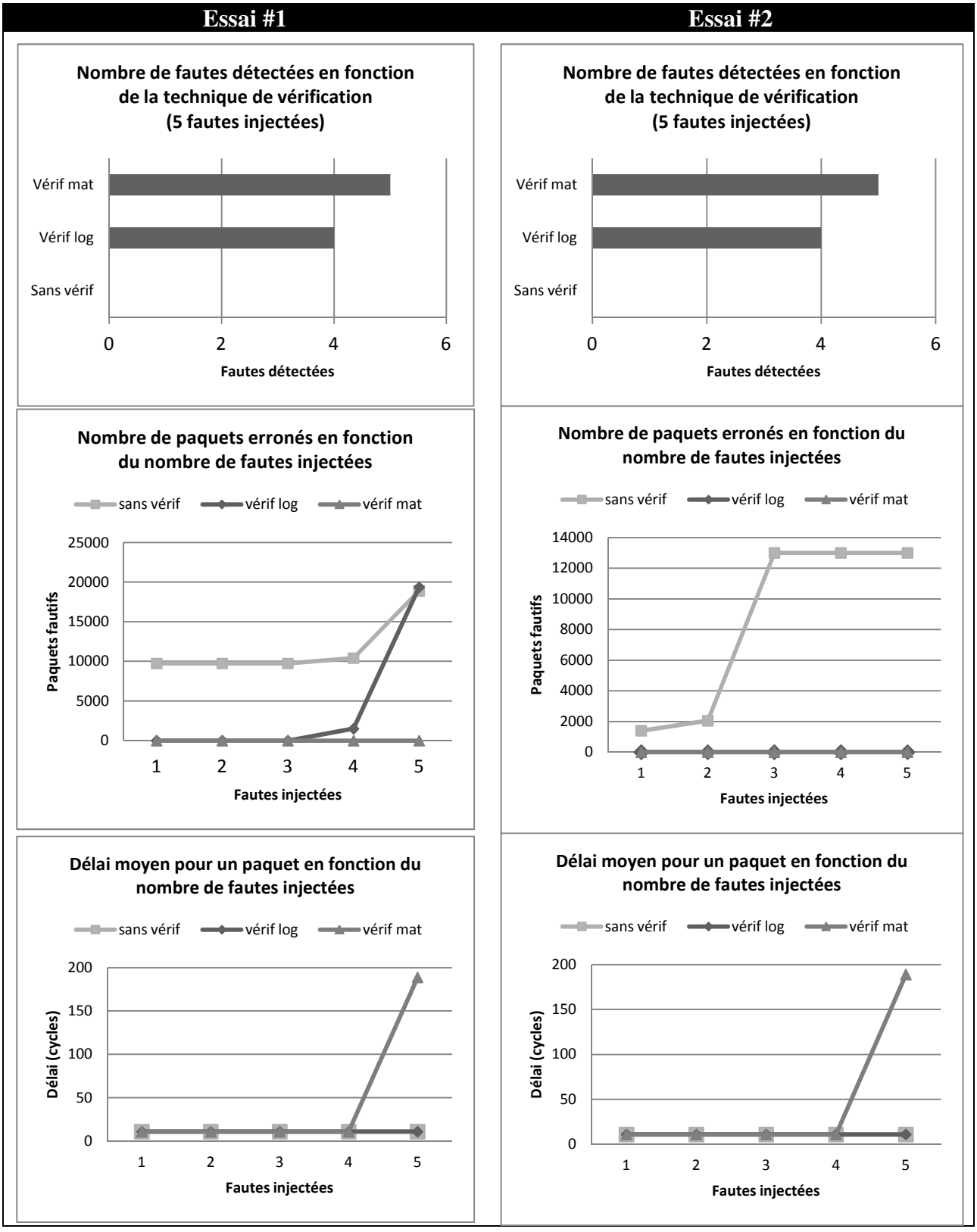
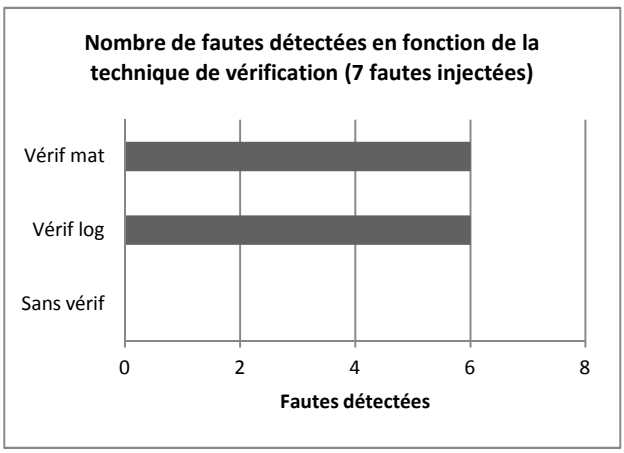
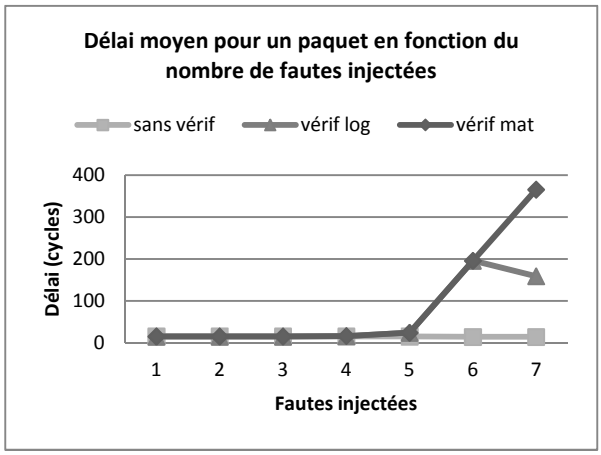


Figure 5-10: Graphiques comparant les techniques de détection avec une configuration 16 nœuds et 3 canaux

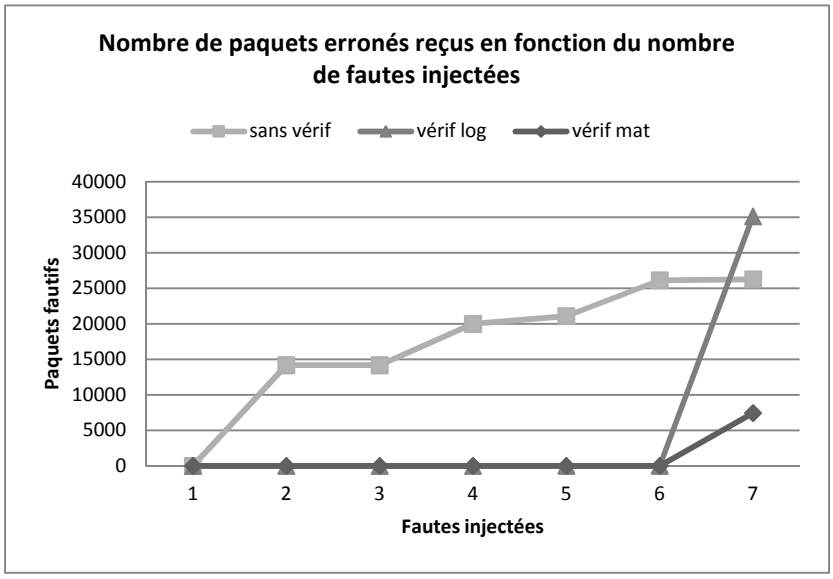
D'autres simulations ont été effectuées avec les mêmes paramètres (détection, reconfiguration et sélection des canaux) pour comparer les techniques de prévention logicielle et matérielle, notamment avec une configuration de 32 nœuds et 4 canaux. Les résultats en découlant sont présentés à la Figure 5-11. Cela a permis de détecter certaines particularités qui pourraient survenir comme le graphique c) ci-dessous le démontre. Dans ce cas-ci, après l'injection de 6 fautes, le trafic étant trop important sur le canal 0, la faute ayant été injectée sur ce canal n'est pas détectée pour la technique de prévention au niveau matériel, car la sélection des canaux se fait toujours en priorisant le canal 0. Par contre, la technique au niveau matériel permet de détecter les fautes concernant les champs de l'en-tête, contrairement à la technique au niveau logiciel.



a) **Nombre de fautes détectées**



b) **Délai moyen des paquets**



c) **Nombre de paquets erronés reçus**

Figure 5-11: Graphiques comparant les techniques de détection pour une configuration 32 nœuds et 4 canaux

5.2.1.3 **Discussion sur les fautes permanentes**

Les sections précédentes ont permis de définir, dans le cas de fautes permanentes sur le réseau, que la détection de fautes à l'aide du CRC était plus efficace que le bit de parité (section *5.1.2*) et que l'algorithme de gestion des fautes utilisant la reconfiguration instantanée obtenait de meilleurs résultats également (section *5.2.1.1*). De façon générale, il a aussi été démontré que la technique de prévention au niveau matériel permettait d'obtenir une moins grande propagation de paquets erronés, comparativement à la méthode au niveau logiciel (section *5.2.1.2*). Par contre, tout cela n'a été analysé que pour des fautes permanentes.

5.2.2 Fautes transitoires

Les sections précédentes ne discutaient que de fautes permanentes. Maintenant, sensiblement les mêmes critères que ceux des sections précédentes seront évalués dans le but de développer la meilleure implémentation possible pour la technique de prévention des fautes au niveau matériel permettant la prise en charge des fautes transitoires également. Ces critères sont : l'algorithme de gestion des fautes, la sélection des canaux pour l'envoi des paquets et le mécanisme de détection des fautes. Le Tableau 5-4 présente les différentes catégories de simulations effectuées en fonction de la configuration du réseau et de la technique de prévention.

Tableau 5-4: Simulations effectuées pour la méthode de prévention au niveau matériel (fautes transitoires)

Technique / Configuration	Priorité de canal		Tourniquet	
	Parité	CRC	Parité	CRC
16 nœuds et 3 canaux	X	X	X	X
32 nœuds et 4 canaux			X	X

5.2.2.1 **Algorithmes de gestion des fautes**

Les résultats de la section *5.2.1.1*, démontrent bien que la technique de reconfiguration instantanée s'avère la plus performante. Cette partie sert plus à appuyer cette conclusion qu'à la démontrer. Les simulations sont effectuées sur une configuration du RoC comportant 16 nœuds et

3 canaux dans le but de comparer l'algorithme de gestion des fautes utilisant la reconfiguration instantanée avec celui utilisant la reconfiguration après 15 occurrences (référence à la section *5.2.1.1*). Toutes les simulations sont exécutées avec la méthode de détection CRC et avec les deux implémentations de sélection des canaux, soit la priorité de canal ou le tourniquet. Dans ce scénario de test, 15 fautes transitoires simples (affectant un seul canal et un seul bit à la fois) sont injectées tout au long de l'application. Les fautes injectées ont une durée de 1 à 50 cycles et un temps de 1 à 3000 cycles peut s'écouler entre deux fautes. Les résultats obtenus sont présentés sur le graphique ci-dessous (Figure 5-12).

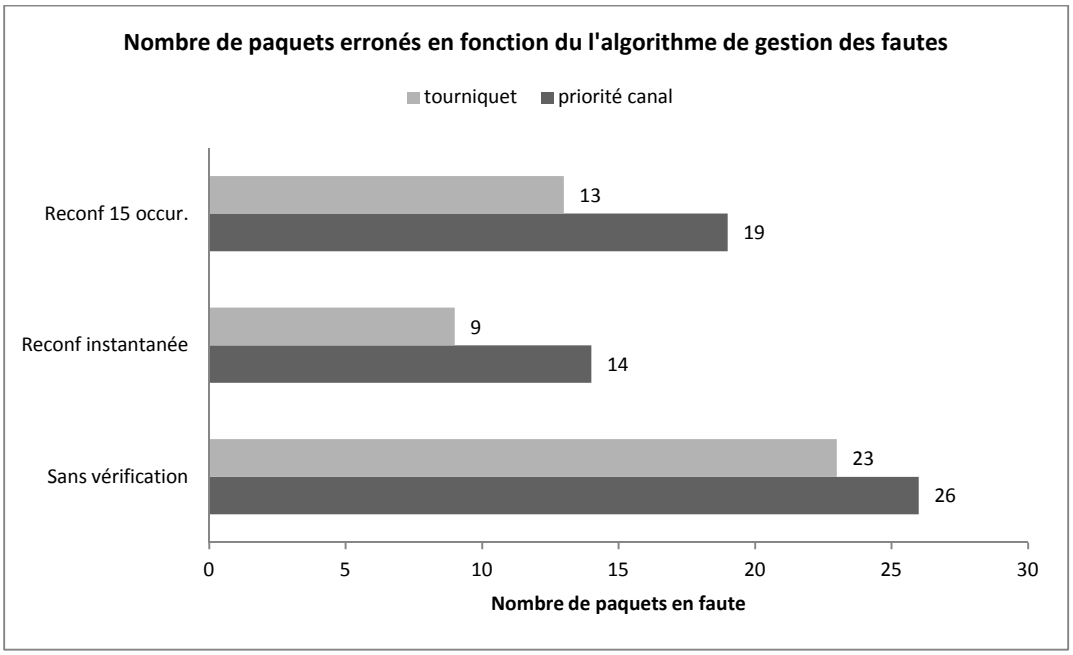


Figure 5-12: Graphique comparant les algorithmes de gestion de fautes avec des fautes transitoires

La technique de reconfiguration instantanée s'avère plus efficace que la technique de reconfiguration après détection de 15 occurrences. Elle permet d'éviter que 25% moins de paquets soient affectés par les fautes présentes. De façon générale, la première technique procède à la désactivation du canal détecté comme étant fautif dès la première apparition d'une faute et le réactive seulement quand le canal a été détecté sans faute *Y* fois consécutives. Cette technique est très performante du côté de la tolérance aux fautes, par contre, elle peut désactiver un canal plus longtemps qu'il ne le faudrait et, de cette façon, ajouter un court délai supplémentaire. La technique de reconfiguration après détection de *X* occurrences, quant à elle, est moins bien en

termes de tolérance aux fautes, car elle laisse quelques paquets se corrompre lors de l'attente des *X* occurrences. Par contre, pour une faute transitoire ne durant qu'un très court instant, moins de *X* cycles par exemple, le canal ne serait pas désactivé, donc pourrait éviter le court délai supplémentaire pouvant être engendré.

5.2.2.2 **Sélection des canaux**

La sélection des canaux pour l'envoi des paquets devient un peu plus critique dans le cas des fautes transitoires. En effet, étant donné que les fautes ne durent qu'un court instant, la détection doit se faire le plus rapidement possible. Deux méthodes différentes sont étudiées dans le but de déterminer la plus efficace : les envois selon la priorité des canaux et les envois selon un principe de tourniquet. La première technique crée un trafic plus élevé sur les premiers canaux et la seconde permet de répartir le trafic plus également sur la totalité des canaux.

Les deux techniques de sélection des canaux ont été simulées par trois scénarios de test différents ayant chacun une configuration 16 nœuds et 3 canaux. Chaque nœud envoie 10 000 paquets à une destination aléatoire. Toutes les fautes injectées sont de type transitoire simple. Le Tableau 5-5 énonce les caractéristiques spécifiques des trois essais exécutés et les résultats obtenus sont présentés sur les graphiques ci-dessous (Figure 5-13 et Figure 5-14).

Tableau 5-5: Paramètres des scénarios de test pour la sélection des canaux

	Nombre de fautes	Durée des fautes (cycles)	Durée entre deux fautes (cycles)	BER max
Essai #1	15	1 à 50	1 à 3000	10^{-5}
Essai #2	30	1 à 100	1 à 1500	10^{-4}
Essai #3	20	1 à 75	1 à 2000	10^{-5}

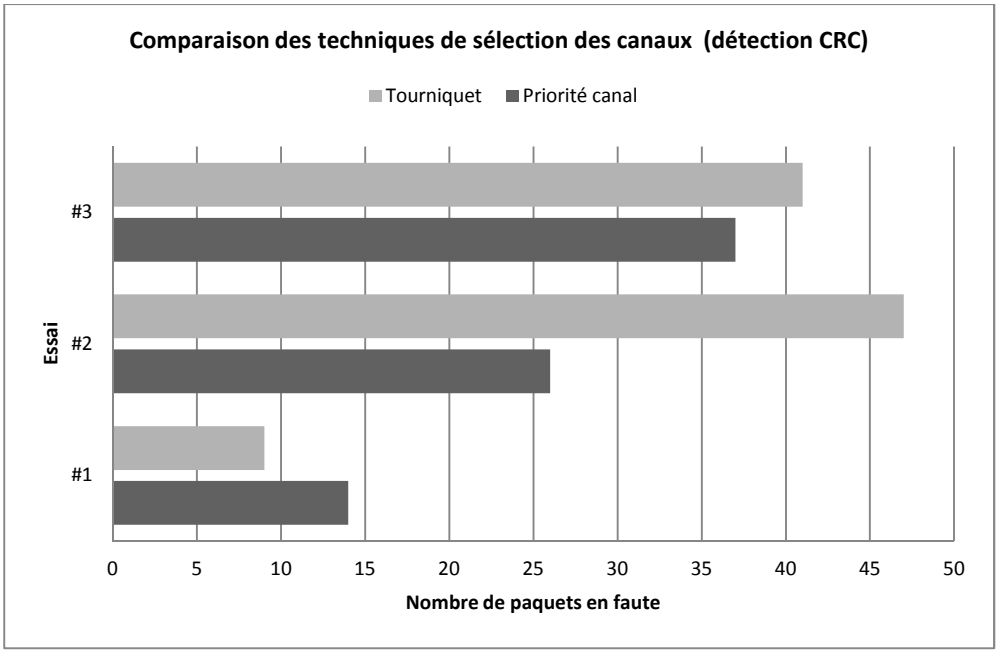


Figure 5-13: Graphique comparant la sélection des canaux avec la technique de détection CRC

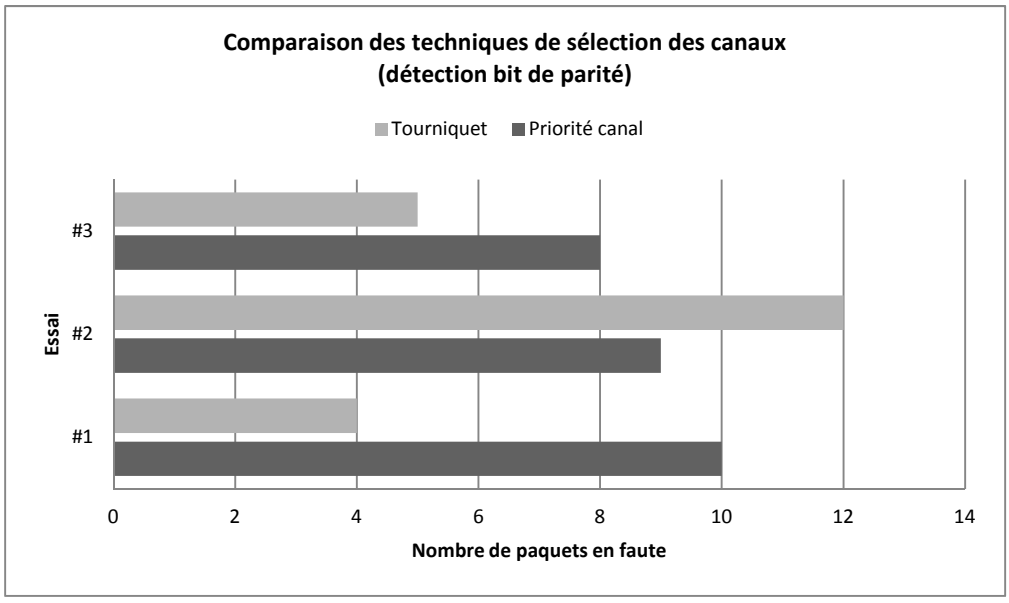


Figure 5-14: Graphique comparant la sélection des canaux avec la technique de parité

Pour le calcul du BER présenté dans le Tableau 5-5, nous avons décidé d'utiliser le pire cas de probabilité d'erreur. Cela arrive lorsque le réseau est surchargé et que tous les canaux ont un trafic similaire. Dans ce cas, peu importe le canal où la faute est présente, il a la même probabilité d'altérer un bit. De plus, considérons que le bit où la faute est présente est toujours altéré.

Bien que les résultats soient plus ou moins concluants, la sélection des canaux selon le principe de tourniquet s'avère être la meilleure technique. Intuitivement, elle permet une répartition plus égale du trafic sur la totalité des canaux du réseau. De cette façon, les chances qu'un canal ne transporte pas de paquet de type applicatif sur un cycle sont plus élevées, ce qui augmente le taux de vérification des canaux. Par exemple, pour la méthode de sélection en priorisant toujours le canal 0, si le trafic est très élevé sur ce canal, il se peut qu'une faute sur le canal 0 ne soit jamais détectée et entraîne la propagation de paquets erronés. Par contre, pour cette même technique, il est certain qu'une erreur sur le quatrième canal aura beaucoup moins de répercussions négatives qu'une sur le premier. Avec le principe de tourniquet, une erreur a le même poids peu importe le canal. Toujours dans une optique de tolérance aux fautes, la sélection des canaux selon un principe de tourniquet permet une meilleure fiabilité.

5.2.2.3 **Technique de détection des fautes**

Tout comme pour la technique de prévention des fautes au niveau matériel, nous tentons de définir la méthode de détection des fautes la plus efficace. Ici, deux mécanismes de base peu coûteux en termes de ressources seront comparés pour la détection, au niveau matériel, des fautes transitoires : le contrôle de parité et le contrôle de redondance cyclique. Comme il a été expliqué précédemment à la section *3.4.2.3 Ajout du module test_receiver*, l'implémentation à l'aide du CRC permet une vérification d'un nouveau canal à tous les deux cycles, tandis que l'implémentation avec le bit de parité permet une vérification à tous les cycles.

Tout comme pour la section précédente, les deux techniques de détection des fautes sont simulées par trois scénarios de test différents ayant chacun une configuration 16 nœuds et 3 canaux. Chaque nœud envoie 10 000 paquets à une destination aléatoire et toutes les fautes injectées sont de type transitoire simple. Le Tableau 5-5 présenté à la section précédente énonce les caractéristiques spécifiques des trois essais exécutés et les résultats obtenus sont présentés sur les graphiques ci-dessous (Figure 5-15 et Figure 5-16).

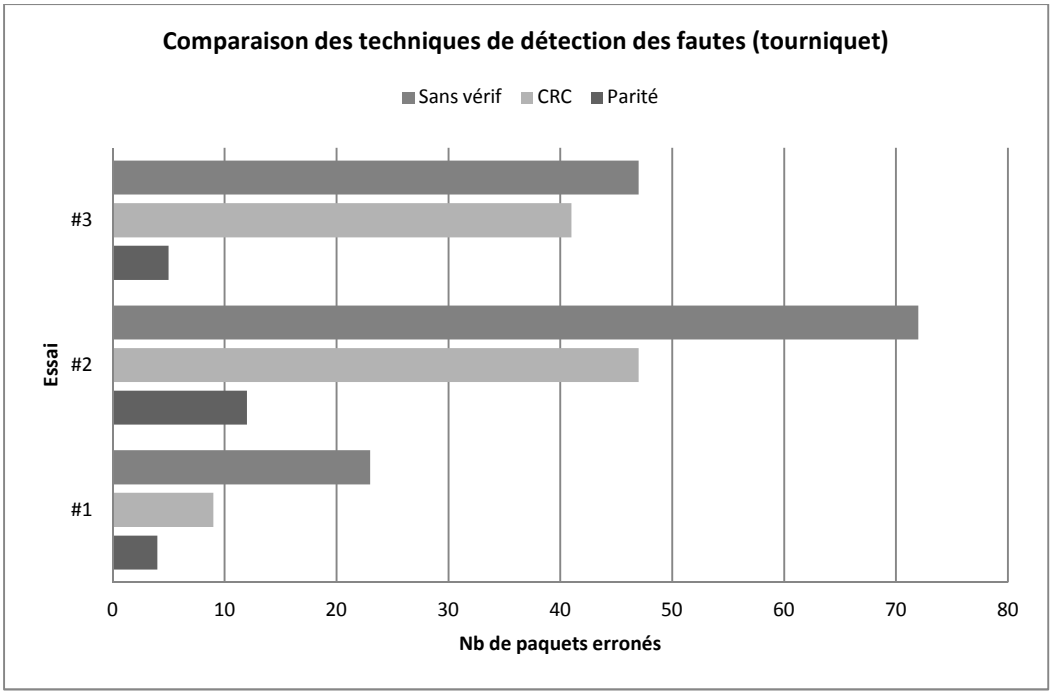


Figure 5-15: Graphique comparant les techniques de détection avec une sélection tourniquet

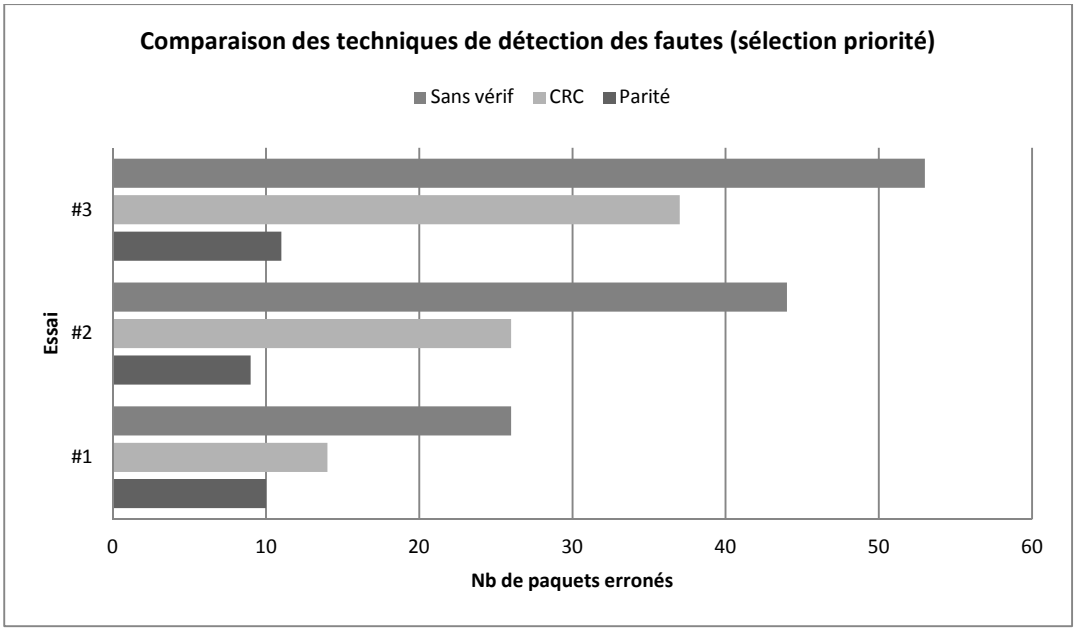


Figure 5-16: Graphique comparant les techniques de détection avec une sélection des canaux par priorité

Bien que ce soit la technique de détection à l'aide du CRC qui ait obtenu la meilleure performance dans le cas de la technique de prévention au niveau logiciel, ici, c'est la technique de contrôle de parité qui permet une meilleure tolérance aux fautes. Cela s'explique principalement par le fait qu'elle permet la vérification d'un nouveau canal à tous les cycles. Le CRC permet une vérification seulement aux deux cycles, car il nécessite un cycle de calcul supplémentaire pour le code de vérification. De plus, la technique de détection avec contrôle de parité nécessite moins de ressources.

5.2.2.4 **Fautes transitoires multiples**

Bien que la détection de fautes simples soit plus critique, ce pourquoi toutes les simulations précédentes ont été effectuées avec cela, il peut arriver que les fautes affectent une région et ne soient pas ciblées sur un bit seulement. Pour évaluer les techniques de détection des fautes dans ce cas, quatre différents scénarios de test sont effectués. Pour les trois premiers, le réseau est configuré avec 16 nœuds et 3 canaux où chaque nœud envoie 10 000 paquets de façon aléatoire. Les fautes injectées ici sont de type transitoire multiple, c'est-à-dire que plusieurs fautes peuvent arriver sur plusieurs bits et sur plusieurs canaux en même temps. De cette façon, nous pouvons également tester la technique de prévention pour s'assurer qu'elle prenne en charge les fautes transitoires qui peuvent simultanément survenir sur plusieurs canaux. Donc, pour les trois premiers scénarios, 32 fautes sont injectées de façon aléatoire sur le réseau tout au long de l'exécution. Les paramètres de durée et de région touchée par les fautes pour les différents essais sont présentés dans le tableau ci-dessous et les résultats correspondant sur le graphique 5-17.

Tableau 5-6: Paramètres des scénarios de test pour les fautes multiples

	Durée des fautes (cycles)	Nombre de bits touchés	BER
Essai #1	1 à 30	1 à 10	10^{-7} à 10^{-4}
Essai #2	1 à 100	1 à 10	10^{-7} à 10^{-3}
Essai #3	1 à 50	1 à 10	10^{-7} à 10^{-4}

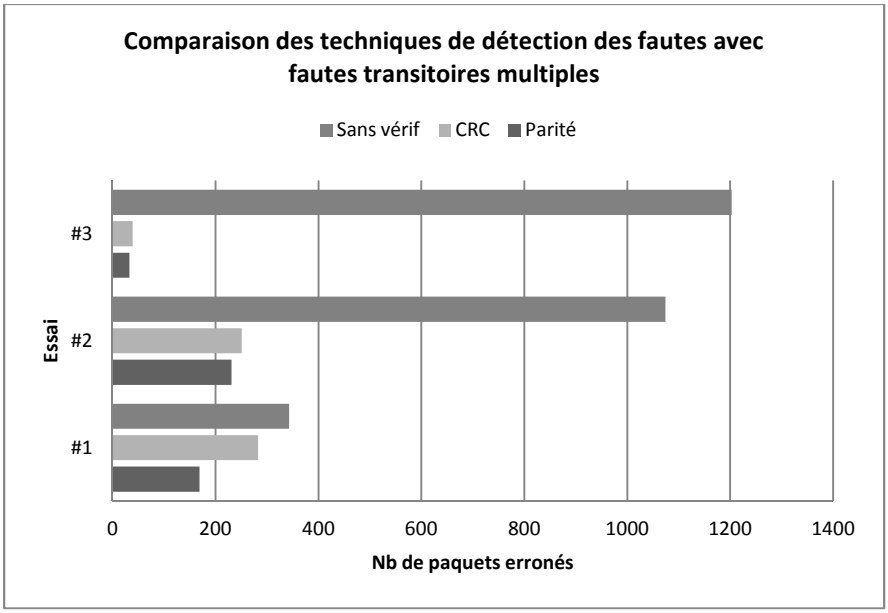


Figure 5-17: Graphique comparant les techniques de détection des fautes pour des fautes multiples

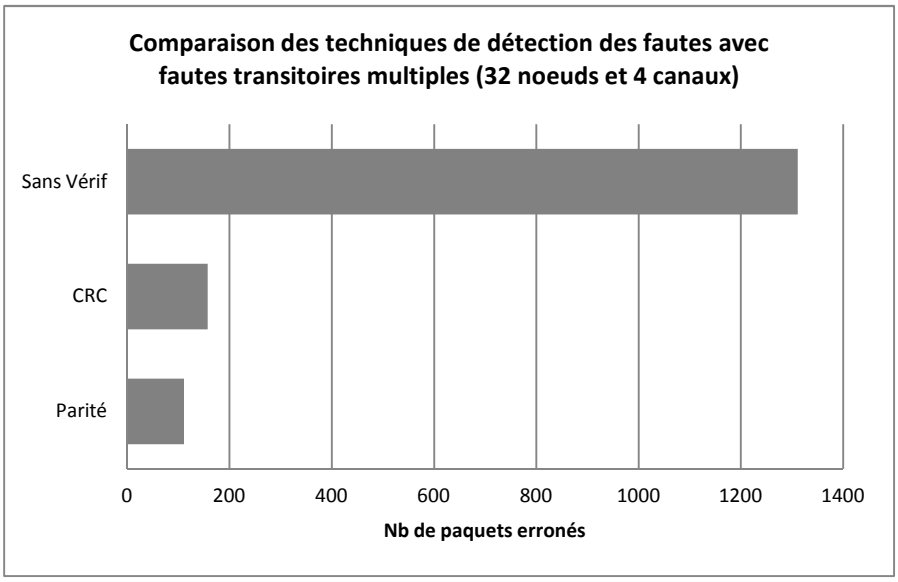


Figure 5-18: Graphique comparant les technique de détection avec une architecture 32 nœuds et 4 canaux

Le quatrième scénario de test, présenté à la Figure 5-18, est exécuté dans le but de corroborer que la technique de détection à l'aide du contrôle de parité s'avère toujours être la meilleure et ce, même avec des fautes multiples. Dans ce cas-ci, une architecture un peu plus lourde est utilisée, soit une configuration avec 32 nœuds et 4 canaux. Les fautes injectées ici sont toujours de type transitoire multiple agissant sur 1 à 10 bits consécutifs et pouvant avoir lieu sur plusieurs canaux simultanément. Pour ce quatrième scénario, 32 fautes durant de 1 à 100 cycles ont été injectées.

5.3 Ressources utilisées

C'est la plateforme de développement ML410 qui a été choisie pour tester le système. Elle est dotée d'un FPGA Virtex 4, XC4VFX60-11FFG1152, de Xilinx. Ce FPGA contient principalement les ressources présentées dans le Tableau 5-7. C'est ce nombre de ressources qui sera pris en compte pour calculer les pourcentages d'utilisation dans le Tableau 5-9. La vérification logicielle implique l'ajout du module *bitmap_error* et des ports s'y rattachant. Les techniques de vérification matérielle impliquent l'ajout des modules *test_generator* et *test_receiver* et sont évaluées avec la même méthode de sélection des canaux, soit le tourniquet. Tous les résultats sont obtenus sans calculer l'ajout du dispositif d'injection de fautes qui est utilisé seulement pour des fins de test.

Tableau 5-7: Ressources disponibles sur le XC4VFX60-11FFG1152

	Slices	*Slice Flip Flops*	*4 Input LUTs*
Nombre	28440	56 880	56 880

Tableau 5-8: Ressources utilisées par les composants de test intégrés aux nœuds

	Mécanisme	**Ressources** *Slices*	*Slice Flip Flops*	*4 Input LUTs*
test_generator	Bit de parité	5	9	1
	CRC-16	26	40	40
test_receiver	Bit de parité	149	101	276
	CRC-16	309	231	591

Tableau 5-9: Ressources utilisées pour un seul nœud dans une configuration 8 nœuds et 3 canaux

Configuration	**Ressources** *Slices*		*Slice Flip Flops*		*4 Input LUTs*	
De base	550	2 %	559	1 %	737	1 %
Vérification logicielle	567	2 %	654	1 %	775	1 %
Vérification matérielle						
Détection avec contrôle de parité	1 041	4 %	801	1 %	1 601	3 %
Détection avec CRC	1 157	4 %	962	1 %	1 765	3 %
Vérification matérielle avec mémoire partagée						
Détection avec contrôle de parité	1 219	4 %	836	1 %	1 909	3 %
Détection avec CRC	1 387	5 %	997	1 %	2 092	4 %

Tableau 5-10: Ressources utilisées pour les différentes techniques de prévention

Configuration	Ressource	Utilisation 8 nœuds et 3 canaux		Utilisation 16 nœuds et 3 canaux	
		Nombre	Pourcentage	Nombre	Pourcentage
De base	*Slices*	6 202	24 %	9 772	38 %
	Slice Flip Flops	7 576	14 %	10 865	21 %
	4 Input LUTs	11 058	21 %	17 561	34 %
Vérification logicielle	*Slices*	6 521	25 %	11 371	44 %
	Slice Flip Flops	7 495	14 %	13 758	27 %
	4 Input LUTs	11 694	23 %	20 366	40 %
Vérification matérielle					
Détection avec contrôle de parité	*Slices*	13 172	52 %	22 387	88 %
	Slice Flip Flops	9 981	19 %	18 741	37 %
	4 Input LUTs	24 755	48 %	42 097	83 %
Détection avec CRC	*Slices*	14 146	55 %	26 793	105 %
	Slice Flip Flops	12 431	24 %	23 638	46 %
	4 Input LUTs	26 917	53 %	48 166	95 %
Vérification matérielle avec mémoire partagée					
Détection avec contrôle de parité	*Slices*	17 737	70 %	34 001	134 %
	Slice Flip Flops	10 532	20 %	19 846	39 %
	4 Input LUTs	33 935	67 %	65 442	129 %
Détection avec CRC	*Slices*	18 744	74 %	36 599	144 %
	Slice Flip Flops	12 979	25 %	24 745	48 %
	4 Input LUTs	36 093	71 %	70 102	138 %

Nous pouvons remarquer que les techniques au niveau matériel sont beaucoup plus coûteuses en nombre de ressources utilisées que celles au niveau logiciel. Cela n'est pas surprenant étant donné que tout est intégré à même le RoC. Également, nous notons que la détection CRC nécessite plus de ressources que le contrôle de parité. Ceci est principalement dû à l'unité de calcul qui doit être intégrée pour le code CRC. Finalement, une autre architecture comportant l'ajout d'une mémoire partagée pour gérer les fautes transitoires multiples au niveau matériel a également été étudiée. Les résultats en termes de ressources utilisées permettent de remarquer que cet ajout s'avère très coûteux pour le peu de paquets erronés qu'il empêche d'être transmis (environ 5 de moins par rapport aux résultats obtenus pour une technique sans utilisation de mémoire partagée, graphique 5-17).

5.4 Discussion

La méthode de prévention au niveau logiciel utilise la détection bout en bout. Dans ce cas-ci, le paquet de test est envoyé par la ressource et est vérifié par la ressource suivante. Par contre, à cause du fait que le paquet soit envoyé à la ressource suivante et que nous connaissons le chemin emprunté, cela peut s'apparenter à une détection connexion à connexion. Les mécanismes de détection étudiés au départ étaient le bit de parité et le CRC-16, car ils permettent un décodage rapide. Par contre, à la suite de quelques évaluations, l'observation a été qu'il fallait vérifier chacun des canaux deux fois à l'aide du CRC-16 pour s'assurer d'un taux de détection d'environ 50%. Si pour obtenir ce taux de détection une double vérification était nécessaire, il était certainement possible de rendre cette dernière encore plus efficace, ce qui a été fait avec la détection utilisant des séquences de bit à « 0 » et à « 1 » qui a permis d'obtenir un taux de détection nettement supérieur, soit environ 90%. Par contre, cette technique ne permet pas de détecter toutes les fautes pouvant survenir sur l'en-tête. De façon générale, la technique au niveau logiciel nécessite moins de ressources que la technique au niveau matériel. Par contre, les probabilités qu'une faute touchant l'en-tête ne soit pas détectée sont plus grandes et cette technique permet la prévention pour les fautes permanentes seulement.

La méthode de prévention intégrée de façon matérielle au RoC utilise une détection connexion à connexion, ce qui implique que les paquets de test sont vérifiés à chaque nœud. Pour les mêmes raisons qu'énoncées précédemment, ce sont des mécanismes de détection simples, dont le bit de parité et le CRC-16, qui ont été utilisés car, en plus, ils nécessitent très peu de ressources. Dans ce cas, c'est le bit de parité qui a obtenu le meilleur taux de détection, car il permet une vérification à chaque cycle d'horloge contrairement au CRC-16. Au niveau de la reconfiguration des canaux pour éviter la propagation des erreurs, c'est la reconfiguration instantanée qui a obtenu les meilleurs résultats. Bien qu'elle consomme plus de ressources matérielles, la technique de prévention au niveau matériel s'avère la plus efficace. Elle s'assure d'une détection sur la totalité de la largeur du chemin de données en plus de prendre en charge les fautes transitoires. Également, elle ne nécessite aucun coût en temps, car la vérification se fait parallèlement à l'exécution de l'application.

CONCLUSION ET TRAVAUX FUTURS

Dans un contexte où les réseaux sur puce ne cessent de gagner en complexité dans le but de connecter toujours plus de ressources et où les transistors les composant ne cessent de se miniaturiser, les systèmes deviennent de plus en plus vulnérables aux fautes. Les fautes pouvant provenir de diverses sources internes ou externes et ayant différents comportements, notamment permanent ou transitoire, il est devenu impératif pour les réseaux sur puce d'être capables de garantir un certain niveau de fiabilité concernant le transfert de données.

L'objectif principal de ce mémoire était donc de concevoir une technique de tolérance aux fautes adaptée au RoC qui s'avérait avoir le meilleur rapport de performance en termes de nombre de fautes évitées, de latence moyenne des paquets, de temps de vérification et de ressources utilisées. Après avoir passé en revue quelques techniques de tolérance utilisées dans les réseaux sur puce, certains concepts de celles-ci ont été retenus. Grâce aux aspects multidimensionnel et bidirectionnel du RoC, il a été possible de développer des méthodes de tolérance aux fautes ne nécessitant aucun canal supplémentaire, ce qui est un attrait très intéressant. De plus, dans le but de ne pas augmenter le chemin critique, toutes les techniques proposées ne vérifient pas les données, mais plutôt des paquets de test qui sont dédiés à cette tâche.

L'optique de prévention des fautes ici a été privilégiée à la correction pure. La correction nécessitant souvent beaucoup de temps de calcul et/ou de ressources, l'idée ici était d'explorer les avantages de la prévention et le taux de fiabilité qu'elle pouvait apporter. Les techniques de prévention développées ici sont toutes régies selon deux principes : la vérification et la reconfiguration. Après la vérification de l'intégrité du chemin de données, si une faute est détectée, il y a reconfiguration des canaux du réseau dans le but d'éviter la propagation des erreurs.

Pour être en mesure de développer la méthode de tolérance aux fautes la plus efficace, il a fallu évaluer quelle méthode de gestion s'avérait la plus performante, soit celle implémentée au niveau logiciel ou celle intégrée directement au RoC de façon matérielle.

La méthode intégrée au niveau matériel, qui utilise le bit de parité comme mécanisme de détection et la reconfiguration instantanée des canaux suite à la détection d'une faute, s'avère être la plus performante. Elle assure une détection sur la totalité de la largeur du chemin de données en plus de prendre en charge les fautes transitoires, ce qui permet d'obtenir le plus bas taux en termes de RPER. Également, elle ne nécessite aucun coût en temps, car la vérification se fait parallèlement à l'exécution de l'application. Par contre, elle nécessite plus de ressources matérielles que la technique de prévention gérée au niveau logiciel.

Dans un environnement où la probabilité de fautes transitoires est très faible, voire quasi nulle, la technique de prévention au niveau logicielle serait à envisager. En effet, elle nécessite beaucoup moins de ressources matérielles, mais il serait possible que certaines fautes ne soient pas détectées au niveau de l'en-tête, ce qui pourrait propager des paquets erronés durant l'exécution. C'est la même chose pour des circuits ayant des défauts de fabrication. Les défauts de fabrication étant habituellement des fautes permanentes, une vérification au niveau logiciel du chemin de données pourrait permettre d'en détecter une partie. Par contre, le problème de l'en-tête perdure. Les circuits pourraient donc être utilisés pour des fonctions non critiques à la place d'être jetés pour mal fonctionnement.

L'une des contributions les plus importantes de ce travail a été de doter le RoC d'un mécanisme de prévention des fautes. Cette technique pourra également être portée sur d'autres réseaux sur puce ayant un aspect multidimensionnel. Elle permet de détecter et de prendre en charge autant les fautes permanentes que transitoires par simple reconfiguration des canaux. En plus de permettre d'éviter de recevoir au-delà de 80% des paquets erronés sans nécessiter de méthode de correction (retransmission ou codes correcteurs), elle ne nécessite pas non plus de canaux supplémentaires et demande un peu moins que le double des ressources utilisées par

l'implémentation de base. Également, ce travail présente une évaluation et une comparaison de différentes méthodes qui peuvent être préférables dans certains cas, même si elles ne s'avèrent pas les plus performantes en tout temps.

Bien qu'une méthode de tolérance aux fautes permettant d'augmenter considérablement le niveau de fiabilité du transfert de données sur le RoC ait été proposée, ce travail ouvre la voie à d'autres problématiques.

Notamment, il serait envisageable de porter la technique développée à d'autres architectures que le RoC ayant des caractéristiques similaires. Selon la méthode de développement de la technique de tolérance, cela semble tout à fait possible de l'intégrer à d'autres réseaux sur puce ayant la particularité d'être multidimensionnels. Par contre, pour s'en assurer, il faudrait mener des essais d'intégration de la technique proposée sur différents réseaux sur puce. Il pourrait également être intéressant d'obtenir des résultats pour des scénarios de simulation représentant des cas un peu plus spécifiques tels que des envois distribués aux nœuds voisins ou le bombardement d'une destination en particulier.

Finalement, certaines optimisations pourraient être apportées à la méthode proposée pour augmenter son efficacité. Tout d'abord, cette technique de prévention des fautes ne permet pas d'atteindre un taux d'erreurs résiduelles de zéro, ce qui serait souhaitable pour assurer une fiabilité sans faille du réseau. Pour y arriver, il serait intéressant de considérer l'intégration de méthodes de correction telles la retransmission ou l'utilisation de codes correcteurs. Également, la technique intégrée au niveau matériel se fait parallèlement au fonctionnement de l'application tout au long de l'exécution. Bien qu'elle ne diminue pas les performances de l'application, la vérification continuelle consomme de l'énergie même dans le cas où aucune faute n'est détectée. Pour cette raison, il serait intéressant d'explorer une façon de procéder à la suspension de la vérification pour être plus efficace d'un point de vue énergétique dans certains cas.

BIBLIOGRAPHIE

Ababei, C., & Katti, R. (2009). Achieving network on chip fault tolerance by adaptive remapping. *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*.(pp. 1-4).

Adriahantenaina, A., Charlery, H., Greiner, A., Mortiez, L., & Zeferino, C. A. (2003). SPIN: a scalable, packet switched, on-chip micro-network. *Design, Automation and Test in Europe Conference and Exhibition, 2003*.(pp. 70-73 suppl.).

Arteris. (2005). A comparison of Network-on-Chip and Busses Consulté le 8 août 2011, Tiré de http://www.design-reuse.com/articles/10496/a-comparison-of-network-on-chip-and-busses.html

Bastos, R. P. (2010). Transient-fault robust systems exploiting quasi-delay insensitive asynchronous circuits. Consulté le 14 septembre 2011, Tiré de http://www.sematech.org

Baumann, R. (2000). Soft error characterization and modeling methodologies at Texas Instruments.(pp. 0043-3283).

Benini, L., & Micheli, G. D. (2006). *Networks on chips: technology and tools*: Elsevier Morgan Kaufmann Publishers.

Bertozzi, D., Benini, L., & De Micheli, G. (2005). Error control schemes for on-chip communication links: the energy-reliability tradeoff. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on, 24*(6), 818-831.

Castagnoli, G., Ganz, J., & Graber, P. (1990). Optimum cycle redundancy-check codes with 16-bit redundancy. *Communications, IEEE Transactions on, 38*(1), 111-114.

Concer, N., Iamundo, S., & Bononi, L. (2009). aEqualized: A novel routing algorithm for the Spidergon Network On Chip. *Design, Automation & Test in Europe Conference & Exhibition, 2009. DATE '09.*(pp. 749-754).

Dally, W., Dennison, L., Harris, D., Kan, K., & Xanthopoulos, T. (1994). The Reliable Router: A reliable and high-performance communication substrate for parallel computers

Douglass Locke, C. (1994). Fault tolerant applications systems; A requirements perspective. *Hardware and Software Architectures for Fault Tolerance*, 21-25.

Ejlali, A., Al-Hashimi, B. M., Rosinger, P., & Miremadi, S. G. (2007). Joint Consideration of Fault-Tolerance, Energy-Efficiency and Performance in On-Chip Networks. *Design, Automation & Test in Europe Conference & Exhibition, 2007. DATE '07*.(pp. 1-6).

Glasser, M. (2009). *Open verification methodology cookbook*: Springer Verlag.

Hadjiat, K., St-Pierre, F., Bois, G., Savaria, Y., Langevin, M., & Paulin, P. (2007). An FPGA Implementation of a Scalable Network-on-Chip Based on the Token Ring Concept. *Electronics, Circuits and Systems, 2007. ICECS 2007. 14th IEEE International Conference on*.(pp. 995-998).

Hamming, R. (1950). Error Detecting and Error Correcting Codes. *Bell System Technical Journal, 26*(2), 147-160. doi:citeulike-article-id:1667687

Hass, K. J. (1999). Probabilistic estimates of upset caused by single event transients.(Vol. 8).

Kariniemi, K., & Nurmi, J. (2005). Fault tolerant XGFT network on chip for multi processor system on chip circuits. *Field Programmable Logic and Applications, 2005. International Conference on*.(pp. 203-210).

Khalili, R., & Salamatian, K. (2005). A new analytic approach to evaluation of packet error rate in wireless networks. *Communication Networks and Services Research Conference, 2005. Proceedings of the 3rd Annual*.(pp. 333-338).

Kologeski, A., Concatto, C., Carro, L., & Kastensmidt, F. L. (2011). Adaptive approach to tolerate multiple faulty links in Network-on-Chip. *Test Workshop (LATW), 2011 12th Latin American*.(pp. 1-6).

Koopman, P., & Chakravarty, T. (2004). Cyclic redundancy code (CRC) polynomial selection for embedded networks. *Dependable Systems and Networks, 2004 International Conference on*.(pp. 145-154).

Lehtonen, T., Liljeberg, P., & Plosila, J. (2007). Online Reconfigurable Self-Timed Links for Fault Tolerant NoC. *VLSI Design, 2007*. doi:10.1155/2007/94676

Li, L., Vijaykrishnan, N., Kandemir, M., & Irwin, M. J. (2003). Adaptive error protection for energy efficiency.(pp. 2): IEEE Computer Society.

Liguo, Z., Huimin, D., & Jungang, H. (2008). ftNoC: A New Architecture of Network on Chip with Fault-Tolerance. *Computing, Communication, Control, and Management, 2008. CCCM '08. ISECS International Colloquium on*.(Vol. 1, pp. 193-197).

Liu, Y., Jungang, H., & Du, H. (2008). A New On-Chip Interconnection Network for System-on-Chip. *Embedded Software and Systems, 2008. ICESS '08. International Conference on*.(pp. 532-539).

Martinez Vallina, F., Jachimiec, N., & Saniie, J. (2007). NOVA interconnect for dynamically reconfigurable NoC systems. *Electro/Information Technology, 2007 IEEE International Conference on*.(pp. 546-550).

Monnet, Y. (2007). *Etude et modélisation de circuits résistants aux attaques non intrusives par injection de fautes*.

MoSys. (2002). MoSys adds soft-error protection, correction to 1-transistor SRAM for 'free'. Consulté le 12 septembre 2011, Tiré de EETimes http://www.eetimes.com/electronics-

news/4093435/MoSys-adds-soft-error-protection-correction-to-1-transistor-SRAM-for-free-

Mukherjee, S. S., Emer, J., & Reinhardt, S. K. (2005). The soft error problem: an architectural perspective. *High-Performance Computer Architecture, 2005. HPCA-11. 11th International Symposium on*.(pp. 243-247).

Murali, S., Theocharides, T., Vijaykrishnan, N., Irwin, M. J., Benini, L., & De Micheli, G. (2005). Analysis of error recovery schemes for networks on chips. *Design & Test of Computers, IEEE, 22*(5), 434-442.

Nunez-Yanez, J. L., Edwards, D., & Coppola, A. M. (2008). Adaptive routing strategies for fault-tolerant on-chip networks in dynamically reconfigurable systems. *Computers & Digital Techniques, IET, 2*(3), 184-198.

Oommen, K., & Harle, D. (2005). Evaluation of a network on chip architecture based on the clockwork routed Manhattan street network using hardware emulation. *Circuits and Systems, 2005. 48th Midwest Symposium on*.(pp. 1625-1628 Vol. 1622).

Ozev, S., Sorin, D. J., & Yilmaz, M. (2007). Low-cost run-time diagnosis of hard delay faults in the functional units of a microprocessor.(pp. 317-324): IEEE.

Palframan, D. J., Nam Sung, K., & Lipasti, M. H. (2011). Time redundant parity for low-cost transient error detection. *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2011*.(pp. 1-6).

Parallel Computer Routing and Communication. In K. Bolding & L. Snyder (dir.). (Vol. 853, pp. 241-255): Springer Berlin / Heidelberg. doi:10.1007/3-540-58429-3_41

Park, D., Nicopoulos, C., Kim, J., Vijaykrishnan, N., & Das, C. R. (2006). Exploring Fault-Tolerant Network-on-Chip Architectures. *Dependable Systems and Networks, 2006. DSN 2006. International Conference on*.(pp. 93-104).

Peterson, W. (1960). Encoding and error-correction procedures for the Bose-Chaudhuri codes. *Information Theory, IRE Transactions on, 6*(4), 459-470.

Pirretti, M., Link, G. M., Brooks, R. R., Vijaykrishnan, N., Kandemir, M., & Irwin, M. J. (2004). Fault tolerant algorithms for network-on-chip interconnect. *VLSI, 2004. Proceedings. IEEE Computer society Annual Symposium on*.(pp. 46-51).

Qiaoyan, Y., & Ampadu, P. (2010). Transient and Permanent Error Co-management Method for Reliable Networks-on-Chip. *Networks-on-Chip (NOCS), 2010 Fourth ACM/IEEE International Symposium on*.(pp. 145-154).

Qiaoyan, Y., & Ampadu, P. (2011). A Dual-Layer Method for Transient and Permanent Error Co-Management in NoC Links. *Circuits and Systems II: Express Briefs, IEEE Transactions on, 58*(1), 36-40.

Samuelsson, H., & Kumar, S. (2004). Ring road NoC architecture. *Norchip Conference, 2004. Proceedings*.(pp. 16-19).

Shamshiri, S., & Kwang-Ting, C. (2009). Yield and Cost Analysis of a Reliable NoC. *VLSI Test Symposium, 2009. VTS '09. 27th IEEE*.(pp. 173-178).

Shivakumar, P., Kistler, M., Keckler, S. W., Burger, D., & Alvisi, L. (2002). Modeling the effect of technology trends on the soft error rate of combinational logic. *Dependable Systems and Networks, 2002. DSN 2002. Proceedings. International Conference on*.(pp. 389-398).

Siguenza-Tortosa, D., & Nurmi, J. (2002). Proteo: a new approach to network-on-chip.

Sobolewski, J. S. (2003). Cyclic redundancy checkla dir. de.), *Encyclopedia of Computer Science* (pp. 476-479): John Wiley and Sons Ltd.

Tanner, R. M. (1984). Fault-Tolerant 256K Memory Designs. *Computers, IEEE Transactions on, C-33*(4), 314-322.

Tavakkol, A., Moraveji, R., & Sarbazi-Azad, H. (2008). Mesh Connected Crossbars: A Novel NoC Topology with Scalable Communication Bandwidth. *Parallel and Distributed Processing with Applications, 2008. ISPA '08. International Symposium on*.(pp. 319-326).

Tezzaron, S. (2004). *Soft Errors in Electronic Memory - A White Paper*.

Wiklund, D., & Dake, L. (2003). SoCBUS: switched network on chip for hard real time embedded systems. *Parallel and Distributed Processing Symposium, 2003. Proceedings. International*.(pp. 8 pp.).

Young Hoon, K., Taek-Jun, K., & Draper, J. (2010). Fault-Tolerant Flow Control in On-chip Networks. *Networks-on-Chip (NOCS), 2010 Fourth ACM/IEEE International Symposium on*.(pp. 79-86).

ANNEXE 1
Calcul du CRC

Pour le calcul d'un CRC, nous avons besoin de deux choses primordiales :

- M(x), le message initial;
- G(x), le polynôme générateur.

Considérons M'(x), le message transmis, comme étant le message initial concaténé au code CRC obtenu, M'(x) = M(x) + CRC. À des fins de vérification, le quotient du message transmis contenant le code CRC divisé par le polynôme générateur devra être nul, M'(x) / G(x) = 0. Le code CRC correspond au reste de la division du message initial par le polynôme générateur voulu.

Voici un exemple avec

- $M(x) = x^3 + x^2 + 1 = 1101$
- $G(x) = x^3 + x + 1 = 1011$

Étant donné que le polynôme générateur est de degré 3, nous devons ajouter 3 bits nuls à la droite du M(x) pour obtenir la largeur du M'(x) désirée.

```
1101000 | 1011
1011    | 1111
--------|
 110000 |
 1011   |
--------|
  11100 |
  1011  |
--------|
   1010 |
   1011 |
--------|
      1 |
```

M'(x) = M(x) + CRC = 1101 + 001 = 1101**001**